DOCUMENT RESUME

ED 052 484 24 CG 006 492

AUTHOR Scott, Heather
TITLE Program Description-HASM: Hierarchical Access Storage Management. Information System for Vocational Decisions.
INSTITUTION Harvard Univ., Cambridge, Mass. Graduate School of Education.
SPONS AGENCY Office of Education (DHEW), Washington, D.C.
REPORT NO PR-34
BUREAU NO BR-6-1819
PUB DATE May 70
GRANT OEG-1-6-061819-2240
NOTE 80p.

EDRS PRICE EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS *Computers, Data Bases, Educational Technology, Information Storage, *Information Systems, *Man Machine Systems, *Occupational Guidance, *Programing Languages

ABSTRACT

Described is a complete and technical description of one component of the Information System for Vocational Decisions (ISVD) total data storage and access system. Both a verbal and a graphic flow chart of its functions within the system are presented. (TL)

ED052484

INFORMATION SYSTEM FOR VOCATIONAL DECISIONS

Project Report No. 34

PROGRAM DESCRIPTION

HASM

Hierarchical Access Storage Management

Heather Scott

This paper was supported in part by Grant No. OEG-1-6-061819-2240
of the United States Office of Education under terms of the
Vocational Education Act of 1963.

Graduate School of Education
Harvard University

May 1970

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

CG 006 492

HASM

Hierarchical Access Storage Management

The total data base for ISVD is divided into records within various files covering SCRIPTS, EDUCATION, OCCUPATIONs, etc. Every time that TRAC needs access to a piece of information, whether it be a script step or a descriptor in a certain record, and every time it needs to store, replace, or modify an existing piece of data, it issues a call to HASM with a pointer set at the specific File Pointer Block (FPB) containing the calling parameters (e.g. record name, descriptor name, etc.).

Whenever a new user logs on the system, TRAC calls a special entry point of HASM which initializes the FPB's, counts, and buffer areas for this user in HP five of the execution area of core, seven half-pages which are paged in and out of core with each user's time slice.

Upon receiving a call from TRAC at the normal entry point, HASM analyzes the function code set by the called in the FPB and branches to the individual routine designed to handle the request.

If the information to be fetched or acted upon in some way is not currently in core, the parameters are transferred from the TRAC FPB to a DASM FPB, and by a superior call, DASM takes over the system until it returns the necessary information (fetch complete or not complete).

There are several different types of functions performed by HASM:
1. Fetch a descriptor with decoding. If the record containing the descriptor is not in core, HASM will call upon DASM to fetch it from the disc, and will then move it to one of the half-pages to be paged in and out with each user. If, after checking through the entire record, the descriptor is not found, an error indicator is set in the FPB and immediate return to TRAC is made.

Otherwise, the $DSTBL record for this file is accessed through DASM and searched through for the descriptor name. If the entry in the $DSTBL indicates that there is a decode table to be used for decoding, the

value string of the descriptor, then that, too, is accessed, the value string is translated into the corresponding decode table value, and this information passed back to TRAC. If there is no entry for the descriptor in the $DSTBL or no decode table entry for the value string, then the value string is returned to TRAC as it is in the descriptor itself.

2. Fetch without decoding. This procedure is quite similar to the above except that the value string of the descriptor is always returned to TRAC without any attempt being made to decode it.

3. Execute a Script Step. This procedure is also similar to a regular 'Fetch', except that each script may have many segments whereas a data record has only one segment. Using a table of contents descriptor at the beginning of each script indicating the upper and lower limits of each segment, the correct segment is fetched into core and then searched for the individual step number. The script step can be requested by TRAC either by its number or simply by asking for "next step" in the FPB, which makes it possible to step through all segments of a script without knowing specific numbers.

4. Binary Vector (as in level one records). In this routine, each bit of a binary vector pointed to by the TRAC FPB, is tested and if it is on (i.e., equal to one), its position in the vector is translated to an integer number. For example, if the third bit within the second byte of the vector equals one, its position is calculated as the 11th bit and therefore its equivalent integer is 11. The integer number is looked up in a decode table named by the $DSTBL entry for this vector name. The text associated with each 'on' bit is moved into a 255-byte buffer, each entry preceded by commas. If the buffer becomes filled before decoding of the vector is completed, an indicator is set in the TRAC FPB and the address of the buffer returned to TRAC. If TRAC needs more decoding done on the binary vector it makes another call and HASM will then continue from that bit in the vector it pointed to when the text buffer became full. If the vector is completely decoded before the buffer is filled, HASM returns to TRAC with the total length of the decoded text in the FPB.

5. Initialize or close-out current user on the system. Initialization consists of fetching via DASM this user's nickname and setting up the FPB for his own personal file. When closing out a user, HASM replaces this personal FPB with the HASM, or system, FPB.

6. Store, insert, replace, or delete a descriptor. When TRAC calls for the storing of a descriptor, HASM first fetches the record and searches through it for this descriptor name.

If the record itself does not exist, HASM will create a new record and assign it the name given by TRAC; it will set up the descriptor as the only descriptor of the record. If the record does exist, but the descriptor does not, HASM adds this new descriptor to the end of the current record and changes the record length by the new descriptor length. If the descriptor in question is already present in the record, HASM replaces its value string with the new value string if the new length is equal to or less than the old length. If not, the present descriptor within the current record is deleted and the new descriptor is set up at the end of the record. In the "insert" function HASM stores a descriptor at the end of the record, providing it is not there at all in the current record. In the "replace" function a descriptor can be stored only if it is there now in the record. In the "delete" function a descriptor is eliminated from the record and if a "delete with fetch" was requested, the value string of the descriptor which has just been deleted is returned via the FPB to TRAC.

When all modifications of a record are complete, HASM calls DASM to store the altered record length and content out on the disc.

HASM is written in assembler language as a module of the Time Sharing and Information Management (TSIM) System and occupies approximately 5 K of core. Also, five of the seven user half-pages at the end of core are allocated for HASM usage:

HP 0	script segment storage
HP 1	decode table storage
HP 4	Data record storage
HP 5	FPBs and user work areas and buffers
HP 6	$DSTBL record storage

FLOW CHART

(Verbal)

x ref.

A. Initialization routine for half-page five

1. Load HASM base registers (GPR2, GPR4) — LØAD
2. Store all remaining registers in half-page five
3. Set up TRAC FPB address table at beginning of HP — FIRST1
   a. Each address is a full-word
   b. A hex 'FF' indicates end of table — A2
4. Set up the eleven TRAC FPB's
   a. Move respective filename into each FPB
   b. Zero out remainder of each FPB
5. Load into register 15 address of current user's login area (from SWS's ECLOG)
6. Set up the four DASM FPB's
   a. Move in full-word address of next FPB into last four bytes of the first three FPB's — A4
   b. Zero out the rest of each FPB
7. Initialize counters
   a. Set current script name to zeros (CURSCRPT)
   b. Set binary decode flag bit to zero (BINBYT)
8. Open "HASMFILE" as current user
   a. a. Set up filename in "NICK" FILE FPB [R.8 has address]
   b. Put function code of hex '66' in VFUNCT
   c. Call DASM through SVC
9. Load back registers previously stored for TRAC
10. Return to TRAC

x ref

B. Regular HASM Entry Point

1. Save registers of calling program — HASM2
2. Load HASM base registers
3. Initialize internal indicators to zero
4. What is function requested of HASM? — A5
   a. Fetch w/ decoding — C1
   b. Fetch w/o decoding — C13
   c. Level 1 record decoding — G1
   d. Execute script step — D1
   e. Store, insert, replace or delete desc. — E1
   f. Initialize new user — H1
   g. Close out current user — I1
   h. DASM functions — F1
   i. All others — 5
5. Set error indicator in caller's FPB to x'0002'
6. Go to C118

C. Fetch with and without decoding

	x ref
1. Access DASM FPB for $DSTBL fetching	FETCH D
2. Is correct DSTBL record in core now?	
yes 11	
no 3	
3. Set STORESAV parameter to address of HP6	
4. Store DSTBL record name in DASM FPB	B1
5. Set segment number count to zero	
6. Call internal subroutine to fetch record into core (M1)	
7. Was record fetched successfully?	
yes 11	
no 8	
8. Zero out VFILNAME of DASM FPB	
9. Set indicator to show bad fetch	
10. Go to 13	
11. Is binary decoding flag set?	B1A
yes 36	
no 12	
12. Is "decode according to DSTBL entry" function requested (VFUNCT = x'62')?	
yes 36	
no 13	
13. Access DASM FPB for data record fetches	FETCHNØD
14. Is record needed in core now?	
yes 19	
no 15	
15. Store Filename and record name in DASM FPB	B2A
16. Set STORESAV parameter to address of HP4	
17. Call internal subroutine to fetch a record into core (M1)	
18. Was record fetched successfully?	
yes 19	
no 121	
19. Is first descriptor requested?	B3
yes 26	
no 20	

	x ref
20. Is next descriptor requested?	
yes 31	
no 21	
21. Call internal subroutine to search through record for descriptor (N1)	
22. Is descriptor in record?	
yes 23	
no 118	
23. Is fetch without decoding requested?	B3A
yes 41	
no 24	
24. Is $DSTBL missing?	
yes 41	
no 25	
25. Go to 36	
26. Access beginning of data record	B4A
27. Add half-word pointing to first descriptor to beg of record	
28. Store result as address of first descriptor of record in caller's FP3	B4B
29. Move name of descriptor into caller's FPB	
30. Go to 23	
31. Access beginning of record	B4C
32. Step up to current descriptor	
33. End of record?	
yes 114	
no 34	
34. Add RAD of current descriptor to beginning address	
35. Go to 28	
36. Access beginning of HP6	B5
37. Step up to 2nd entry of record	
38. Is this a $DSTBL?	
yes 39	
no 114	

		x ref
39.	Search through table for descriptor name	B5A
40.	Is descriptor entry there?	
	yes 40A	
	no 54	
40A.	Go to 59	
41.	Are we doing binary decoding?	B6
	yes 42	
	no 44	
42.	Turn off binary decode flag	
43.	Go to 114	
44.	Are we decoding according to DSTBL entry?	
	yes 45	
	no 47	
45.	Set error indicator in caller's FPB to show no entry in DSTBL for this descriptor	
46.	Go to 118	
47.	Zero out DASM FPB error indicators	
48.	Store address of descriptor in VCDESP of caller FPB	
49.	Store value string count in FPB VØRCT	
50.	Store address of value string in FPB VØRPT	
51.	Zero out error indicator in FPB	RETURN
52.	Restore all registers	
53.	Return to calling program	
54.	Set STORESAV parameter to address of HP6	B6A
55.	Is last segment of $DSTBL in core now?	
	yes 41	
	no 56	
56.	Access DASM FPB for $DSTBL fetches	
57.	Call internal subroutine to get next segment of a record into core (D54)	
58.	Was fetch successful?	
	yes 58A	
	no 41	
58A.	Go to 36	

	x ref
59. Are we decoding according to entry in $DSTBL?	B7
yes 63	
no 60	
60. Are we doing binary decoding?	
yes 61	
no 41	
61. Is decode table name count within DSTBL entry equal to zero?	B7A
yes 41	
no 62	
62. Go to 64	
63. Is this a binary entry?	B7B
yes G1	
no 61	
64. Access DASM FPB for decode table record fetches	B8
65. Same filename as last fetch?	
yes 66	
no 67	
66. Is a segment of needed decode table in core now?	
yes 76	
no 67	
67. Clear out DASM FPB	B8A1
68. Move decode table name from $DSTBL to DASM FPB	
69. Make record name count an even number and add null byte if necessary	
72. Set sequence # count to zero	B8A2
73. Set STORESAV parameter to address of HP1	
74. Call internal subroutine to fetch a record into core (M1)	
75. Was fetch successful?	
yes 76	
no 121	
76. Access beginning of decode table record	B8A
77. Step up to beginning of table segment header	
78. Save table segment header in TABSAVE	

		x ref
79.	Is code byte equal to 2?	
	yes 80	
	no 114	
80.	Are we doing binary decoding?	
	yes G6	
	no 81	
81.	Are we decoding according to entry in $DSTBL?	
	yes 82	
	no 87	
82.	Save address and length of value string to be decoded	
83.	Clear CØDESAVE buffer	
83A.	Go to 89	
84.	Move zeros into VØRCT of FPB	B8A3
85.	Store address of decode buffer in FPB VØPPT	
86.	Go to 51	
87.	Access address of descriptor to be decoded	B8A4
88.	Save length of value string	
89.	Clear out CØDESAV buffer	
90.	Is value string count equal to zero?	B8A6
	yes 84	
	no 91	
91.	Move text from descriptor into CØDESAV buffer	
92.	Restore decode table pointers	B8AC
93.	Step up to code width in table segment header	
94.	Is code from descriptor within limits of this decode table?	B8B
	yes 95	
	no 114	
95.	Initialize pointers for checking limits of each segment	
96.	Is code from descriptor within limits of this segment?	LØØP2
	yes 99	
	no 97	
97.	Add one to segment count	
98.	Have we checked all segments?	
	yes 114	
	no 98A	

		x ref
98A.	Go to 96	
99.	Is segment needed in core now? yes 100 no 102	
100.	Step through entries in decode table	B9
101.	Can we find a match? yes 101A no 114	
101A.	Go to 105	
102.	Store segment number to be fetched in DASM FPB	B9B
103.	Set STØRESAV parameter to address of HP one	
104.	Call internal subroutine to get specific record segment into core (D54)	
104A.	Go to 100	
105.	Step up to count of decoded text	B10
106.	Are we doing binary decoding? yes G41 no 107	
107.	Is text count equal to zero? yes 109 no 108	
108.	Move decoded text to DECØDE buffer in HP five	
109.	Store count of text in caller's FPB VØRCT	B11
110.	Store address of DECØDE buffer in FPB VØRPT	
111.	Set current descriptor pointer in FPB to address of uncoded text in data record	
112.	Restore register of calling program	
113.	Return to calling program	
114.	Set VERRØR in caller's FPB to x'0800'	ERR4
115.	Are we doing level one record decoding (binary)? yes 116 no 118	
116.	Add 1 to bit counter	
117.	Go to G50	

	x ref
118. Zero current descriptor pointer in caller's FPB	ERR4A
119. Restore registers of calling program	
120. Return to calling program	
121. Move VERRØR from DASM FPB to FPB of caller	DASMERR
122. Go to 118	

	x ref	com.
D. Execute a Script Step		
1. Access Script File Pointer Block (FPB)	EXECUTE	
2. Same script name as last request?		
no 3		
yes 26		
3. Store name of new script in CURSCRPT		
4. Set up script FPB for call to DASM	C1	
5. Set STØRESAV parameter to address of user HP zero		
6. Call DASM to fetch record into HP indicated by STØRESAV parameter		
7. Was fetch to disc completed successfully		
yes 8		
no C121		
8. Is the first script step being requested?	C2	
yes 23		
no 9		
9. Is next sequential step being sought?	C3	
yes 10		
no 28		
10. Access DASM's script FPB		Get
11. Look at address of current step (VCDESP)		next
12. Is this step last one in segment?		script
yes 21		step
no 14		
14. Add half-word link address to current descriptor pointer	C6	
15. Store address of new current step in VCDESP		
16. Move value string (text) of descriptor to buffer in HP5 (DECØDE)	C7A	
17. Store value string count into VØRCT of script FPB		

		x ref	com.
18.	Store address of DECØDE buffer area in VØRPT of script FPB		
19.	Zero out error flags in FPB (VERRØR)		
20.	Return to calling program		
21.	Set "STØRESAV" to address of HPO		
22.	Call internal subroutine to fetch next segment of script into core (D54)		
23.	Load starting address of record into work register (WØRK1)	C5	
24.	Add first link address to starting address to skip over table of contents		
25.	Go to D14		
26.	Is this a request for first step of script? yes 27 no 9	C4	
27.	Is first segment in core? yes 23 no 4		
28.	Set segment number count to zero	C15	Get
29.	Look at $$descriptor in record (Table of Contents)		script
30.	Is step requested higher than lowest step in script? yes 31 no 52		step by spec. name Check
31.	Skip up pointer to next entry in Table of Contents		through table
32.	Is step no. higher than highest entry for this segment? yes 44 no 33	C16	of contents descrip.
33.	Store current pointers in table of contents		

	x ref	com.
34. Set STØRESAV parameter to HP zero		
35. Store segment no. wanted in FPB		
36. Call internal subroutine to fetch specific segment (D54)		
37. Call internal subroutine to search through record for specific name (N1)		
38. Was the script step found? yes 39 no 41		
39. Go to internal subroutine to point to value string count of script step (01)		
40. Go to 16		
41. Restore current place in table of contents	C20	continue with tab. of contents
42. Zero out error indicators in FPB		
44. Add one to segment number of script	C16A	check
45. Skip up pointers to next entry in table		
46. Have we reached end of table? yes 47 no 32		
47. Set error indicator in script FPB for "script step not found"		
48. Go to C-118		
52. Skip to next pair of entries in table	C18	
53. Go to 44		
54. Is segment requested already in core ? yes Return to caller no 55	C9	Get right segment into core
55. Step up pointer to end of record name		
56. Is first segment of record in core? yes 57 no 63		
57. Move in segment number to end of record name		

		x ref	com.
58.	Add 2 to length of record name	C10	
59.	Store new length in DASM FPB (VØRCT)	C10A	
60.	Move in new segment number as current segment	C10B	
61.	Call internal subroutine to fetch a record into core (M1)		
62.	Was record fetched successfully?		
	yes Return to caller		
	no C121		
63.	Is request for first segment of a record?	C12	
	yes 69		
	no 64		
64.	Is there a null byte at end of record?		
	yes 67		
	no 65		
65.	Store in new segment no. at end of record name		
66.	Go to 60		
67.	Store in segment no. followed by null byte at end of record name		
68.	Go to 60		
69.	Move zeros into end of record name to old segment no.	C12A	
70.	Subtract two from count of recname length		
71.	Go to 59		

	x ref
E. Store, Insert, Replace, and Delete Descriptors	
1. Access DASM FPB for storage & fetching of records	STØRED
2. Is the record to be modified already in core?	S100
yes 7	
no 3	
3. Set STØRESAV parameter to address of HP4	S150
4. Move name of record to be fetched into DASM FPB	
5. Call subroutine to fetch a record (MM1)	
6. Was fetch successful?	
yes 7	
no 68	
7. Call subroutine to search for a descriptor within a record (N1)	S200
8. Was the descriptor found?	
yes 9	
no 48	
9. Has the "insert" function been used? (x'5C')	
10. Has "delete" function (with or without fetching) been used? (x'5F' or x'60')	
yes 33	
no 11	
11. Call routine to step pointer up to value string count (Ø1)	
12. Is length of new value string greater than old length?	
yes 13	
no 15	
13. Is new length just 1 byte longer?	
yes 14	
no 22	
14. Is old length count an odd number (.'. extra null byte at end)?	
yes 15	
no 22	
15. Go to 18	

		x ref
16.	Set VERRØR in caller FPB to indicate error in calling HASM	S225
17.	Go to C118	
18.	Move new value string count into descriptor	S250
19.	Is count equal to zero? yes 63 no 20	
20.	Move new value string into descriptor	
21.	Go to 63	
22.	Can new descriptor fit into the record? yes 23 no 26	S300
23.	Is there a desc. with same name there now? yes 28 no 24	
24.	Call routine to build new descriptor in buffer (E28)	
25.	Go to 54	
26.	Set error indicator in FPB (VERRØR) to show there is no room for new desc.	S325
27.	Go to C118	
28.	Make first half-word of descriptor zeros to show it is new "last descriptor" of record	S350
29.	Store descriptor name count and name in buffer	
30.	Store descriptor value string count into buffer	
31.	Is total length of new descriptor greater than 255 bytes? yes 26 no 31A	
31A.	Move text of value string into buffer	
32.	Return to calling routine	
33.	Is this an empty record? yes 52 no 34	S400
34.	Access old descriptor within the record	
35.	Store RAD of old descriptor into RAD of descriptor preceding it	

x ref

36. Was old descriptor originally the last descriptor?
    yes 37
    no 38
37. Store address of previous descriptor in 2nd half-word of record to show it is new "last descriptor"
38. Is call for a "delete only" function (x'5F')? S500
    yes 63
    no 39
39. Is call for a "delete with preceding fetch" function (x'60')?
    yes 43
    no 40
40. Is RAD pointing to first descriptor now at zeros?
    yes 69
    no 41
41. Call routine to build new descriptor in buffer (E28)
42. Go to 54
43. Call routine to point to value string CT of descriptor (Ø1) DELFETED
44. Move value string count into VØRCT of caller's FPB
45. Move value string text into "decode" buffer
46. Store address of buffer in VØRPT of caller's FPB
47. Go to 63
48. Is call for a "replace descriptor" function (x'5D')? S700
    yes 52
    no 49
49. Is call for one of "delete" functions (x'5F' or x'60')?
    yes 52
    no 50
50. Is this record empty of valid descriptors (first half-word equal to zero)?
    yes 69
    no 51
51. Go to 22

		x ref
52.	Set error indicator in VERRØR of caller's FPB	S750
53.	Go to 75	
54.	Access beginning of record in HP4	S800
55.	Access RAD pointing to last descriptor within record (2nd half-word of record)	
56.	Add RAD to beginning address of record	
57.	Update new RAD by length of descriptor that is currently last	
58.	Align new RAD to half-word boundary	
59.	Store new RAD in 2nd half-word of record (pointing to new "last descriptor")	
60.	Replace RAD of former "last desc." with RAD of new descriptor	
60A.	Add length of new descriptor to length of total record	
61.	Store new record length in DASM FPB	
62.	Move newly-constructed descriptor from buffer to end of record in HP4	S850
63.	Set function code (VFUNCT) of DASM FPB to x'0E' to show storage function request	S875
64.	Zero out (VERRØR) error indicator in DASM FPB	
65.	Call DASM (through SVC)	
66.	Was "store" function completed successfully? yes 67 no P1	
67.	Go to 75	
68.	Is call for a "store" function (x'5E')? yes 69 no P1	S900
69.	Call routine to create new descriptor in buffer (E28)	S925
70.	Access beginning of HP4	
71.	Set RADs at beginning of record to point to the first descriptor	
72.	Add length of new descriptor and length of first two half-words together	

	x ref
73. Store result as length of new record in DASM FPB	
74. Go to 62	
75. Restore TRAC registers	RETURNS
76. Return to calling program (TRAC)	

x ref

F. DASM Function

1. Access DASM FPB
2. Call DASM with SVC
3. Restore registers of calling program
4. Return to calling program

DASMFUN

	x ref	comm.
G. Level One Record Decoding		
1. Initialize buffer length count to zero (BUFCNT = 0)	BINARY	
2. Set flag to indicate binary decoding in progress (BINBYT = 1)		
3. Clear out text buffer area		
4. Is function code x'62' (decode acc. to entry in $DSTBL)?		
yes C61		
no 5		
5. Is correct $DSTBL record in core?		
yes C36		
no C4		
6. Is this first call to decode binary string?	BIN 1	
yes 12		
no 7		
7. Divide bit counter by 8 (to find no. of bytes completed)		
8. Have we already reached end of string?		
yes 19		
no 9		
9. Lead into work register beginning address of string		
10. Add # of bytes already checked through to beginning address		
11. Go to 22		
12. Point to beginning of vector	BIN 1B	
13. Load in no. of bytes to be decoded		
14. Set bit counter to one		
15. Is current byte all zeros?	BIN 2	
yes 16		
no 21		

		x ref	comm.
16.	Add 8 to the bit counter		
17.	Up the address pointer to next byte of string		
18.	Have we reached the end of the string?		
	yes 19		
	no 15		
19.	Zero out VARG in calling FPB to indicate all decoding has been done	BIN 2C	
20.	Go to 52		
21.	Save current pointers	BIN 2D	
22.	Is next bit of the current byte on?	BIN 3	
	yes 25		
	no 23		
23.	Add 1 to bit counter		
24.	Have we finished this byte?		
	yes 17		
	no 22		
25.	Call internal subroutine to decode current on bit (or bit counter - 38)		
26.	Go to 22		
38.	Convert bit number from binary to decimal	BIN 4	
39.	Make bit number same length as entries in decode table		
40.	Go to C92		
41.	Is buffer already full (256 byte max)	BIN 5	
	yes 20		
	no 42		
42.	Will new decoded text fit into buffer?		
	yes 43		
	no 50		
43.	Add length of text to current buff. length		
44.	Insert comma before text	BIN 5A	

	x ref	comm.
45. Is length count = zero?		
yes 47		
no 46		
46. Move decoded text from decode table into buffer		
47. Update pointer to end of text	BIN 5B	
48. Restore old pointers showing current place in binary string		
49. Return to calling routine (26)		
50. Convert bit counter to binary	BIN 6	
51. Store bit counter in user's FPB		
52. Move address of text buffer into FPB	BIN 7	
53. Store in length of decoded text		
54. Turn off decoding indicator (BINBYT = 0)		
55. Return to calling program		

	x ref	comm.
H. Initialize new user FPB		
1. Access DASM FPB	INITUSER	
2. Store in record name to be fetched		
3. Set "STØRESAV" parameter to address of HP4		
4. Call internal subroutine to fetch record into HP indicated by STØRESAV (M1)		
5. Was fetch successful?		
yes 6		
no _		
6. Go to internal subroutine to search for descriptor requested (N1)		
7. Was descriptor found?		
yes 8		
no 8		
8. Go to internal subroutine to point to value string of current descriptor (Ø1)		
9. Move value string of descriptor to DECØDE buffer in HPfive		
10. Store length of value string in caller's FPB		
11. Store address of buffer DECØDE in same FPB		
12. Access inquirer's individual FPB		
13. Set function code to indicate "close out old user" (x'67')		
14. Call DASM (pointing to inquirer's FPB)		
15. Set up descriptor name as 'NICKNAME' in FPB		
16. Call internal subroutine to search for this descriptor		
17. Was descriptor found?		
yes 18		
no —		

	x ref	comm.
18. Go to internal subroutine to point to value string count of this descriptor (Ø1)		
19. Move in value of NICKNAME descriptor as first half of inquirer's filename in that FPB		
20. Set function code in inquirer's FPB to indicate "initialize for new user" (x'66')		
21. Call DASM (pointing to inquirer's FPB)		
22. Return to calling program		

	x ref	comm.
I. Close out current user		
1. Get DASM FPB pointer to current FPB	ENDUSER	
2. Call DASM		
3. Move "HASM" into first half of file-name, replacing old user's nickname		
4. Return to calling program		

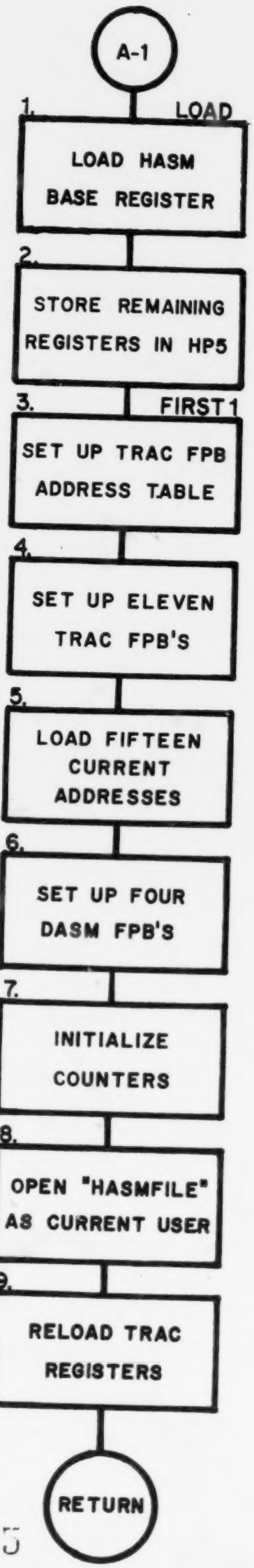
	x ref	comm.
M. Routine to call DASM to fetch a record		
1. Zero out VERRØR1 in DASM FPB	AA1	
2. Set DASM function to x'0B' (fetch record)		
3. Move in filename from caller's FPB to DASM FPB		
4. Call DASM (SVC DASM)		
5. Was DASM action successful?		
yes ≤ 6		
no Return to caller		
6. Access record where brought into core by DASM		
7. Move record into HP indicated by STØRESAV parameter	SAVE	
8. Set DASM function code to x'32' (release HP)		
9. Call DASM (SVC)		
10. Set address in DASM FPB to HP indicated by STØRESAV		
11. Return to calling routine		

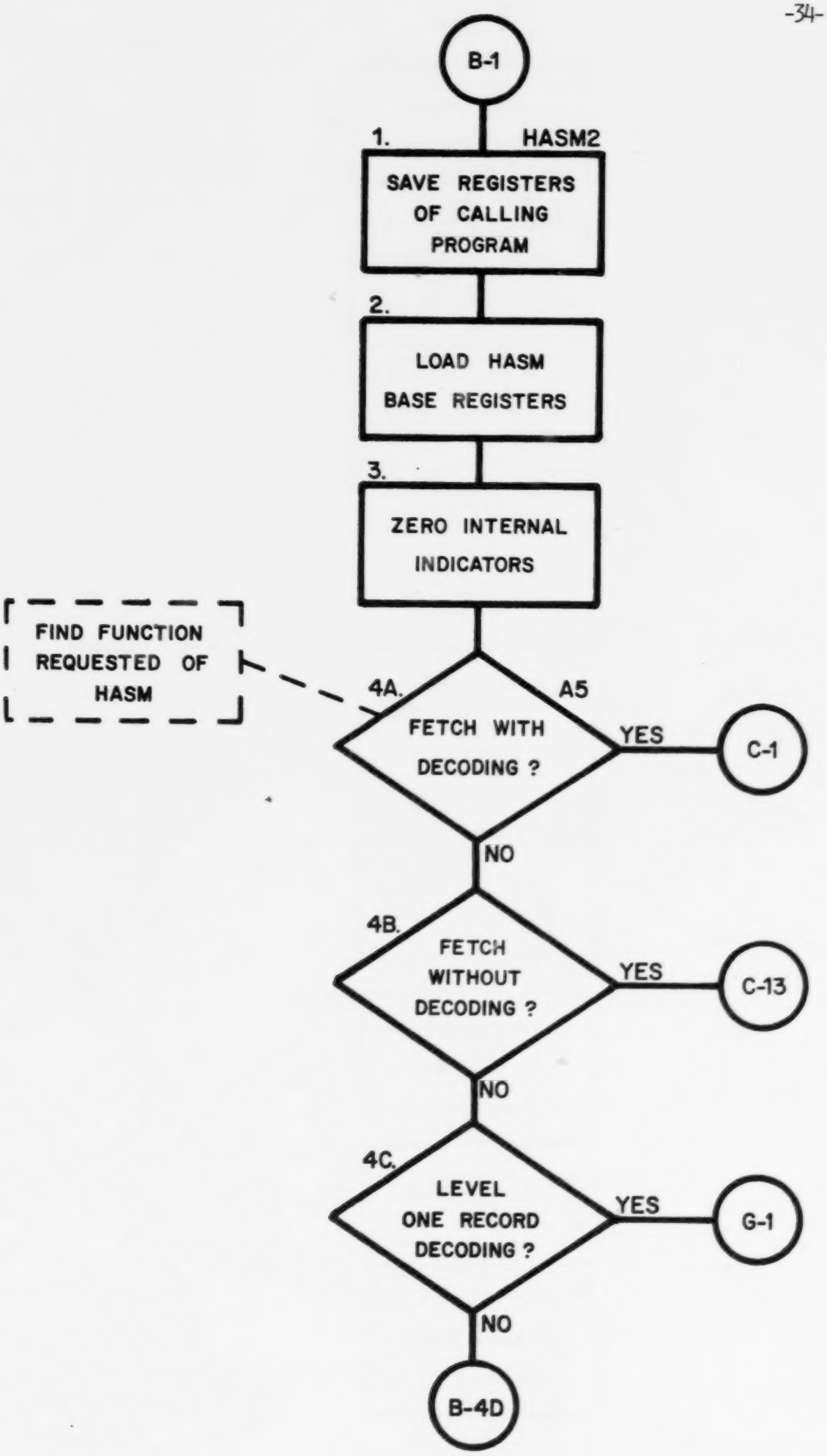
	x ref	comm.
N. Scan record for descriptor to be fetched		
1. Load address of beginning of record into work register	BB1	
2. Link to next descriptor in record by adding relative address displacement to beginning address	BB2	
3. Is this the name of the descriptor requested?		
yes 8		
no 4		
4. Is this the last descriptor in this record? (zeros in relative address halfword)		
yes 11		
no 5		
5. Link to next descriptor		
6. Is this descriptor the one requested?		
yes 8		
no 7		
7. Is this last descriptor?		
yes 11		
no 2		
8. Store address of descriptor into caller's FPB	BB3	
9. Return to calling routine (through r.14)		
11. Set VERRØR in caller's FPB to x'0040' to indicate that descriptor was not found	BB5	
12. Return to calling routine (through r.14)		

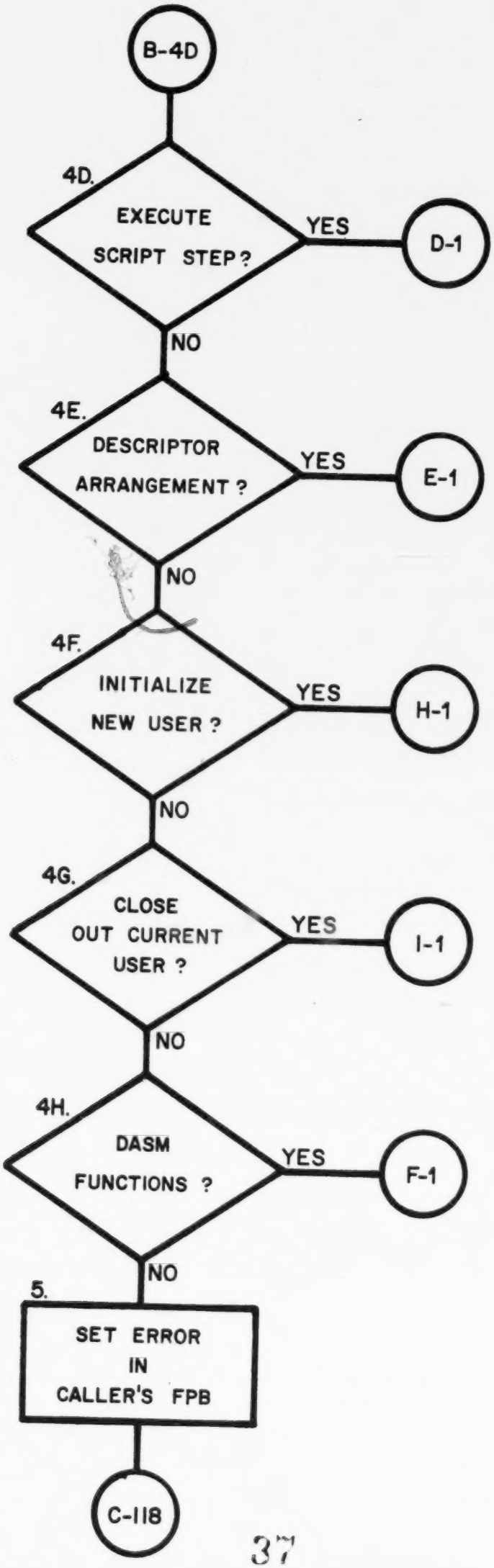
	x ref	comm.
Ø1 Routine to set current desc. pointer and work reg. to val string count		
1. Access address of beginning of descriptor (DESCAD)	CC1	
2. Add length of descriptor name to beginning address		
3. Add 3 more to length for RAD and descriptor name length		
4. Store address of descriptor in current descriptor pointer of caller's FPB		
5. Return to calling routine w. work reg. pointing to val string count of descriptor		

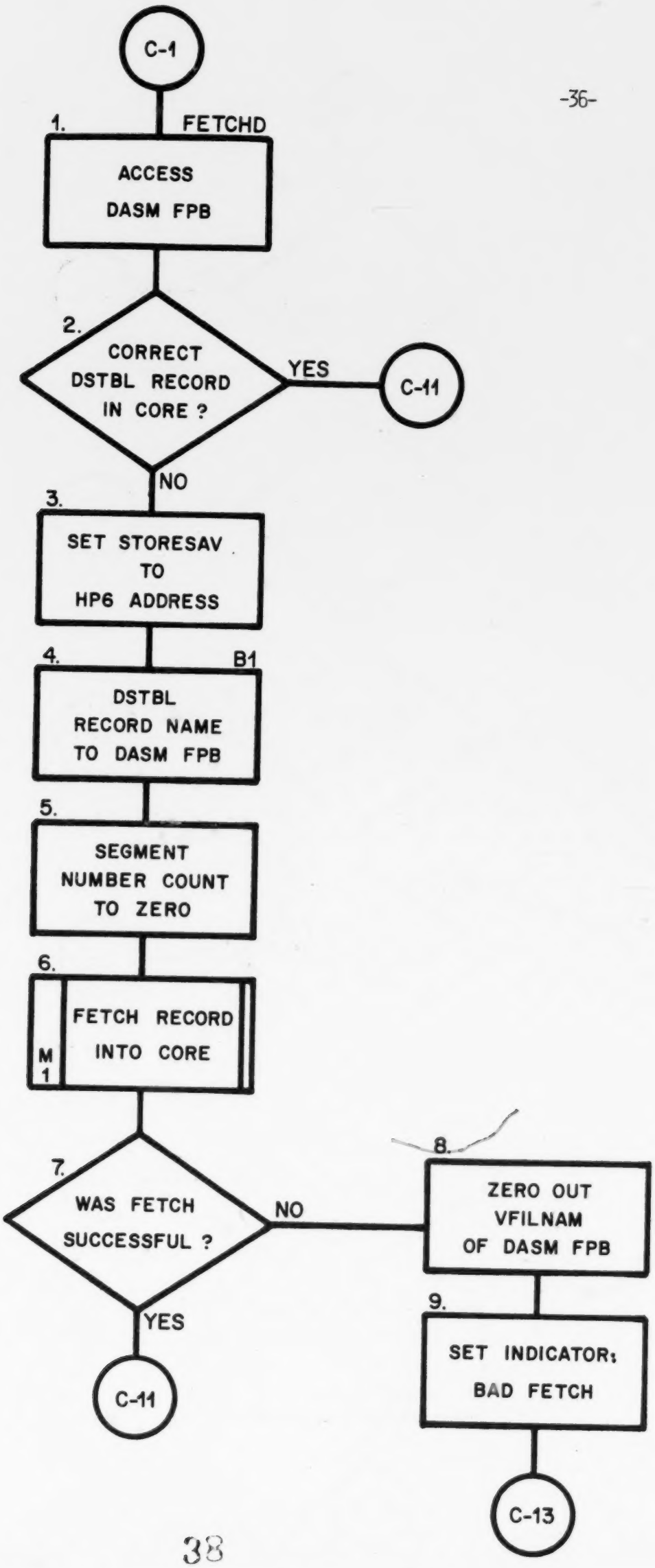
FLOW CHART

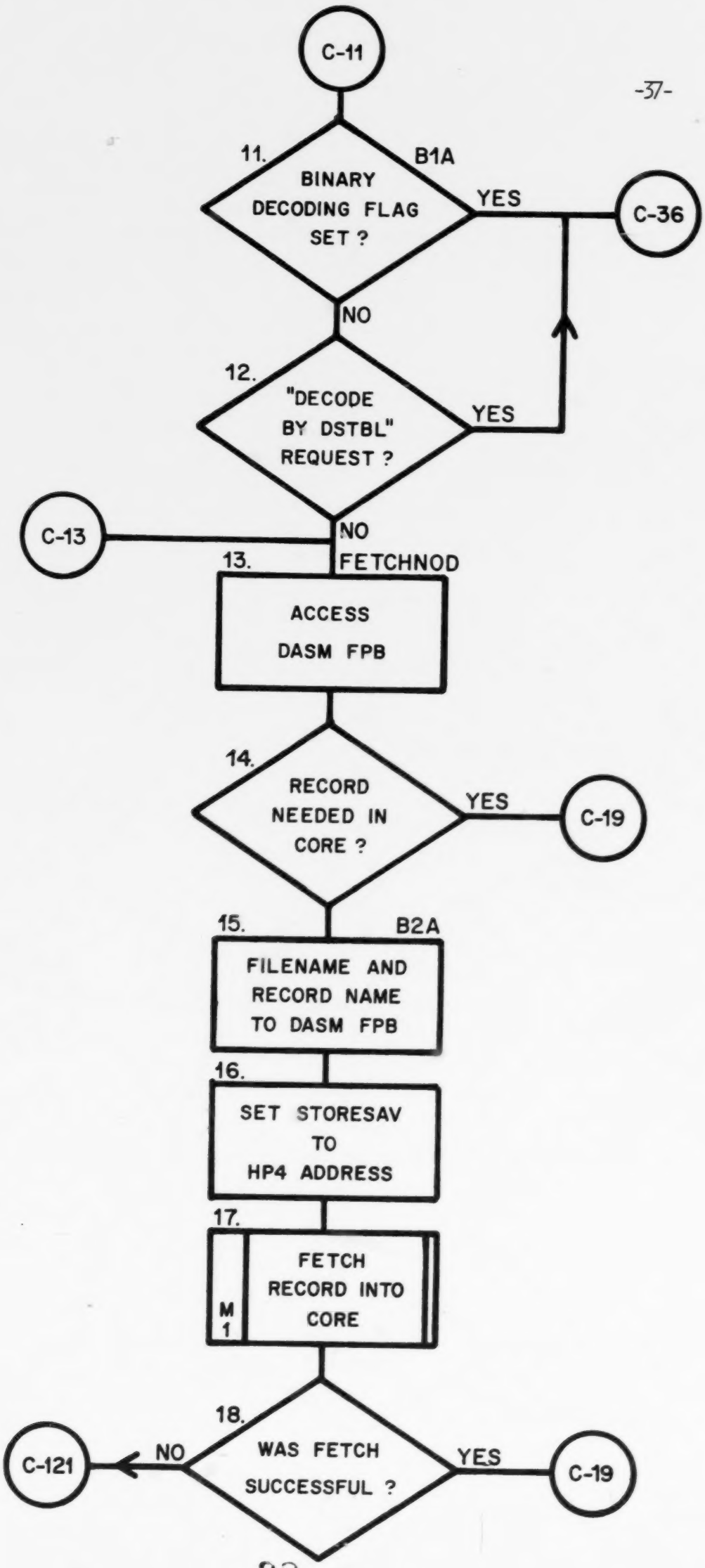
(Graphic)

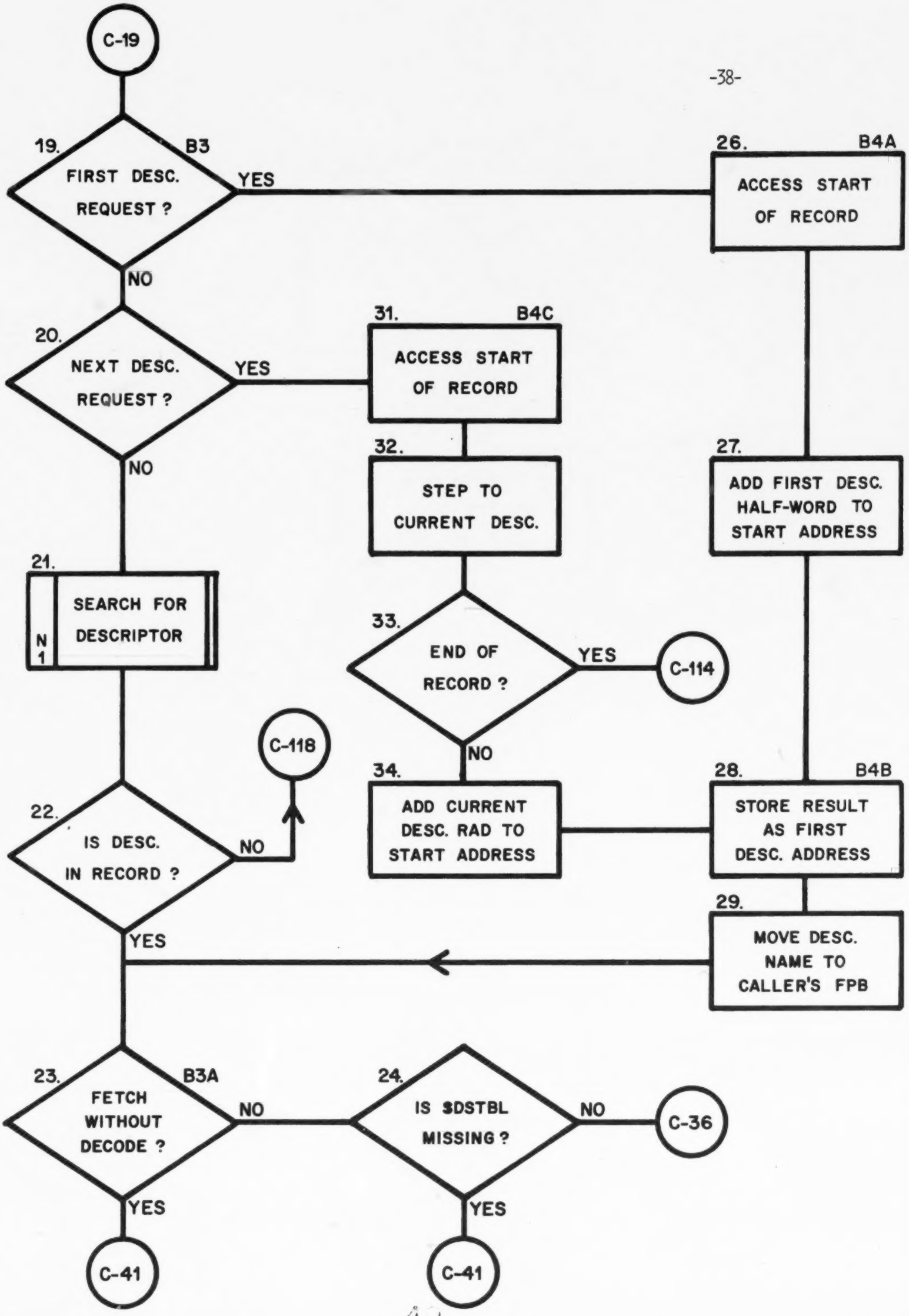

A-1
1.
LOAD
LOAD HASM BASE REGISTER
2.
STORE REMAINING REGISTERS IN HP5
3.
FIRST1
SET UP TRAC FPB ADDRESS TABLE
4.
SET UP ELEVEN TRAC FPB'S
5.
LOAD FIFTEEN CURRENT ADDRESSES
6.
SET UP FOUR DASM FPB'S
7.
INITIALIZE COUNTERS
8.
OPEN "HASMFILE" AS CURRENT USER
9.
RELOAD TRAC REGISTERS
RETURN

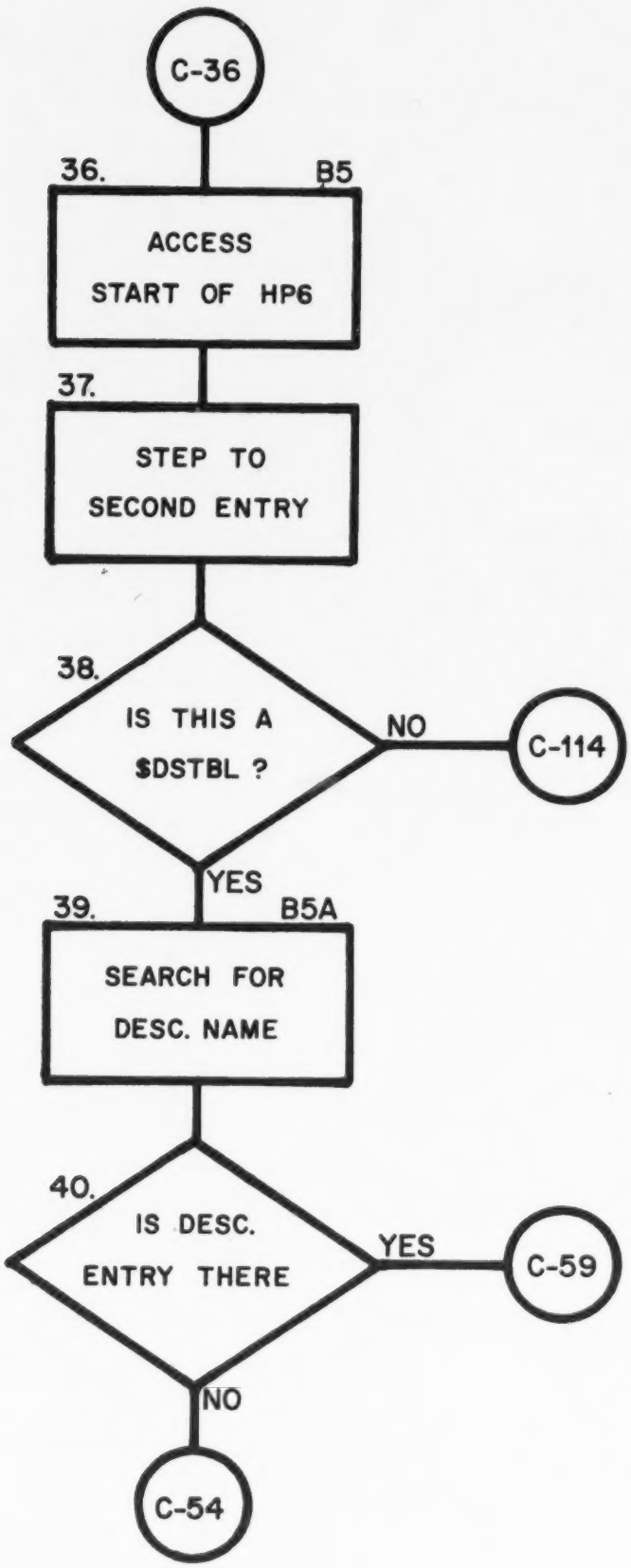

B-1
1.
HASM2
SAVE REGISTERS OF CALLING PROGRAM
2.
LOAD HASM BASE REGISTERS
3.
ZERO INTERNAL INDICATORS
FIND FUNCTION REQUESTED OF HASM
4A.
A5
FETCH WITH DECODING ?
YES
C-1
NO
4B.
FETCH WITHOUT DECODING ?
YES
C-13
NO
4C.
LEVEL ONE RECORD DECODING ?
YES
G-1
NO
B-4D

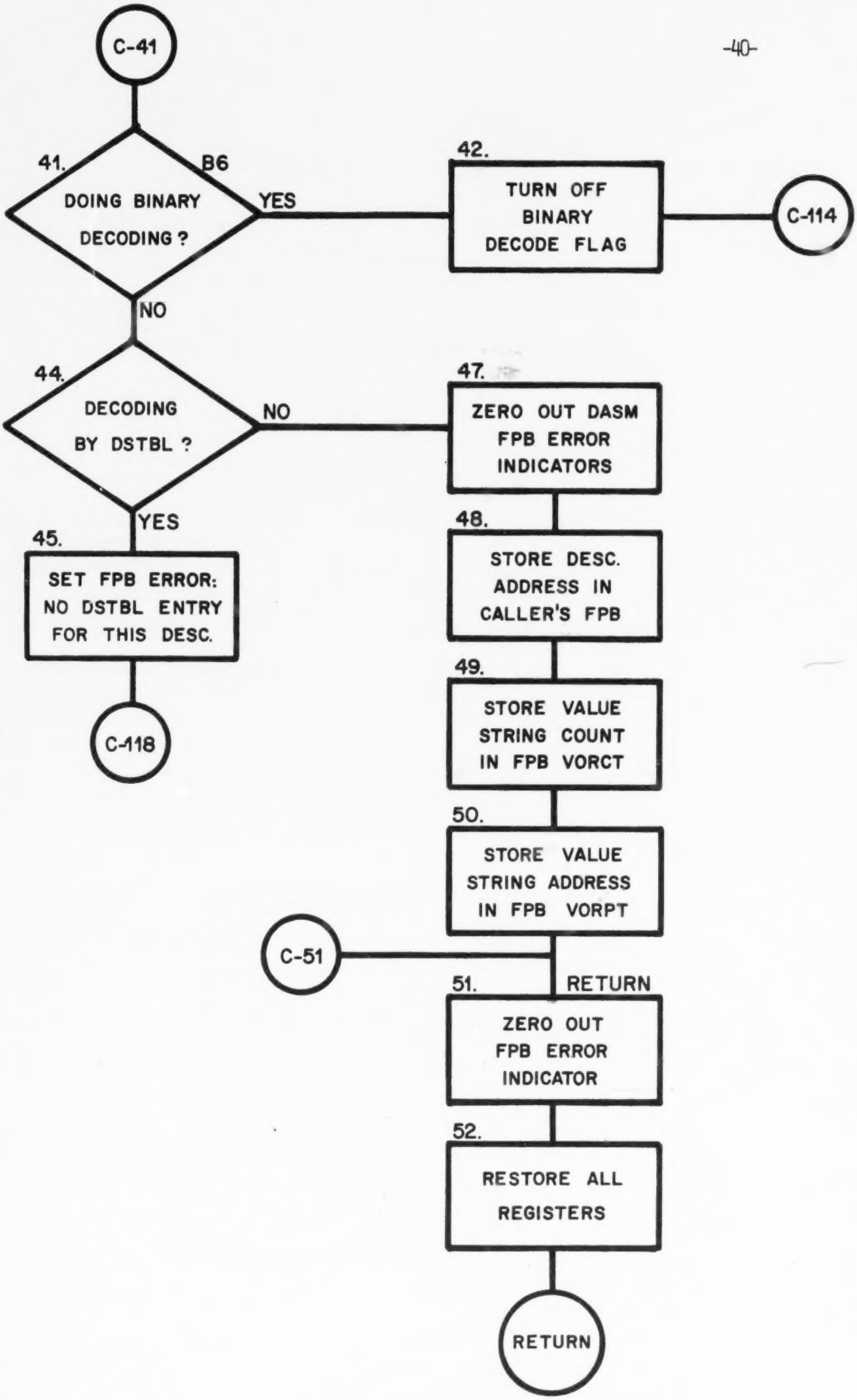

B-4D
4D.
EXECUTE SCRIPT STEP?
YES
D-1
NO
4E.
DESCRIPTOR ARRANGEMENT ?
YES
E-1
NO
4F.
INITIALIZE NEW USER ?
YES
H-1
NO
4G.
CLOSE OUT CURRENT USER ?
YES
I-1
NO
4H.
DASM FUNCTIONS ?
YES
F-1
NO
5.
SET ERROR IN CALLER'S FPB
C-118

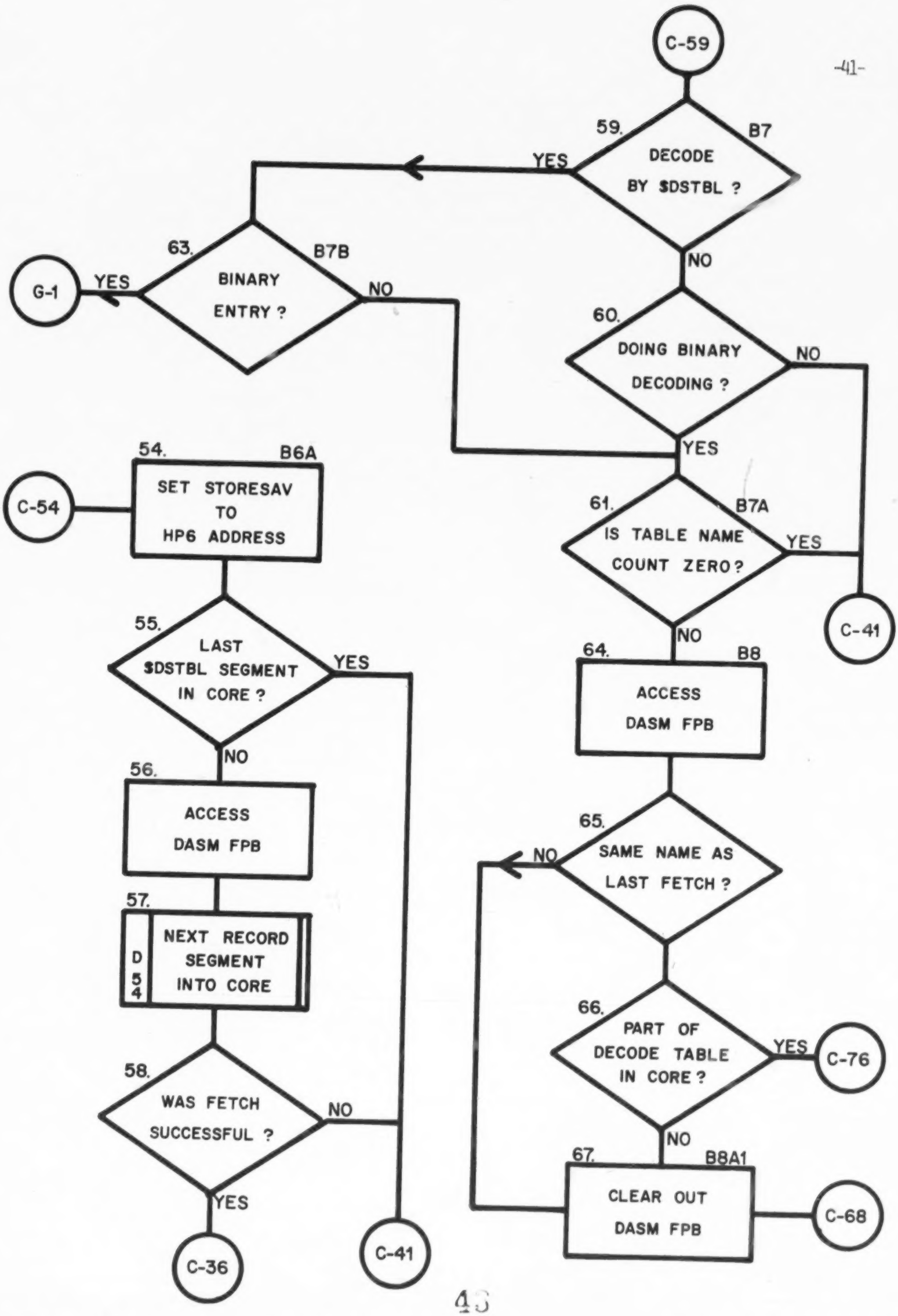

C-1
1.
FETCHD
ACCESS
DASM FPB
2.
CORRECT
DSTBL RECORD
IN CORE ?
YES
C-11
NO
3.
SET STORESAV
TO
HP6 ADDRESS
4.
B1
DSTBL
RECORD NAME
TO DASM FPB
5.
SEGMENT
NUMBER COUNT
TO ZERO
6.
M
1
FETCH RECORD
INTO CORE
7.
WAS FETCH
SUCCESSFUL ?
NO
8.
ZERO OUT
VFILNAM
OF DASM FPB
9.
SET INDICATOR:
BAD FETCH
YES
C-11
C-13

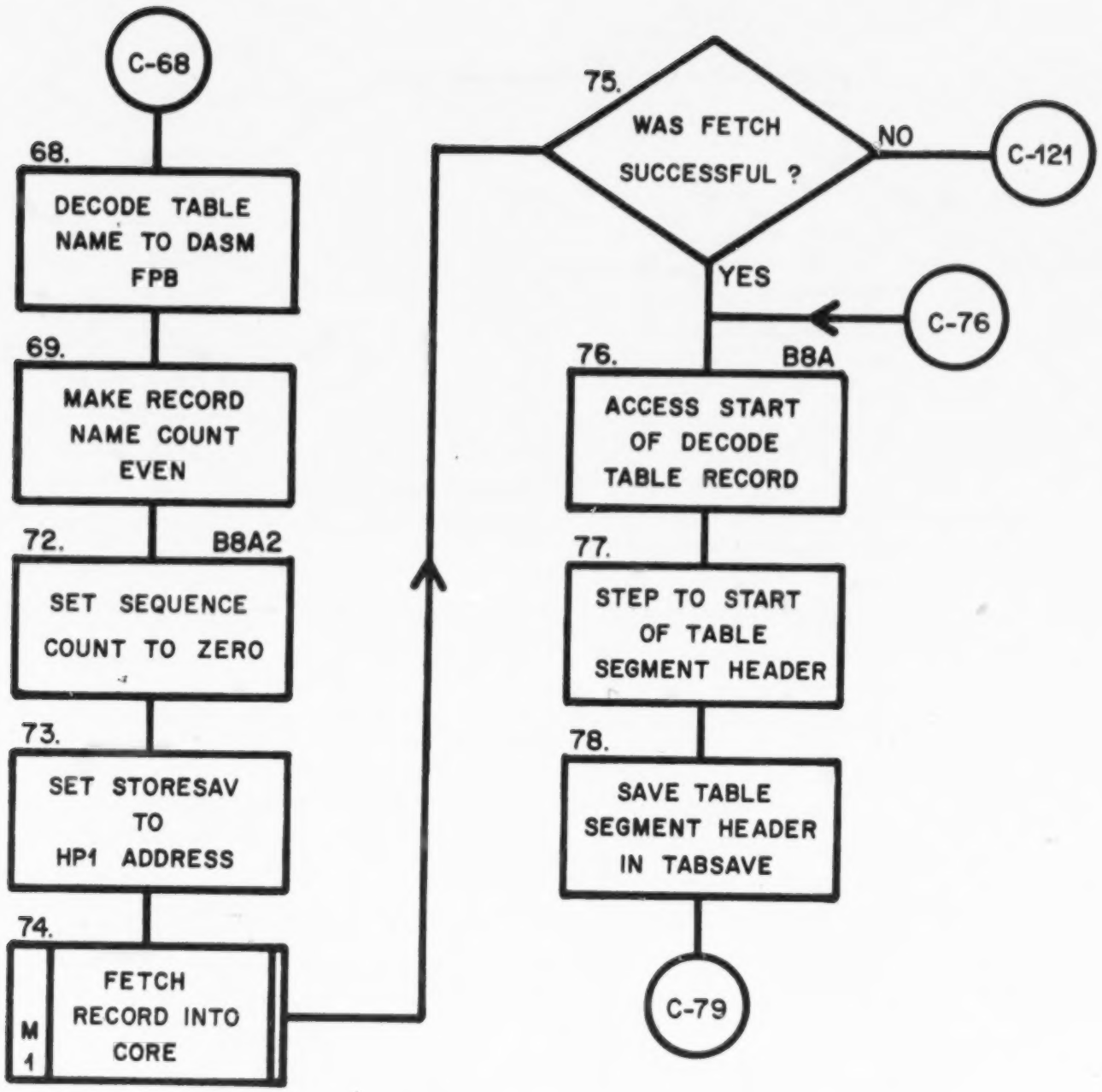

C-11
11.
B1A
BINARY DECODING FLAG SET ?
YES
C-36
NO
12.
"DECODE BY DSTBL" REQUEST ?
YES
C-13
NO
13.
FETCHNOD
ACCESS DASM FPB
14.
RECORD NEEDED IN CORE ?
YES
C-19
15.
B2A
FILENAME AND RECORD NAME TO DASM FPB
16.
SET STORESAV TO HP4 ADDRESS
17.
M 1
FETCH RECORD INTO CORE
18.
WAS FETCH SUCCESSFUL ?
NO
C-121
YES
C-19

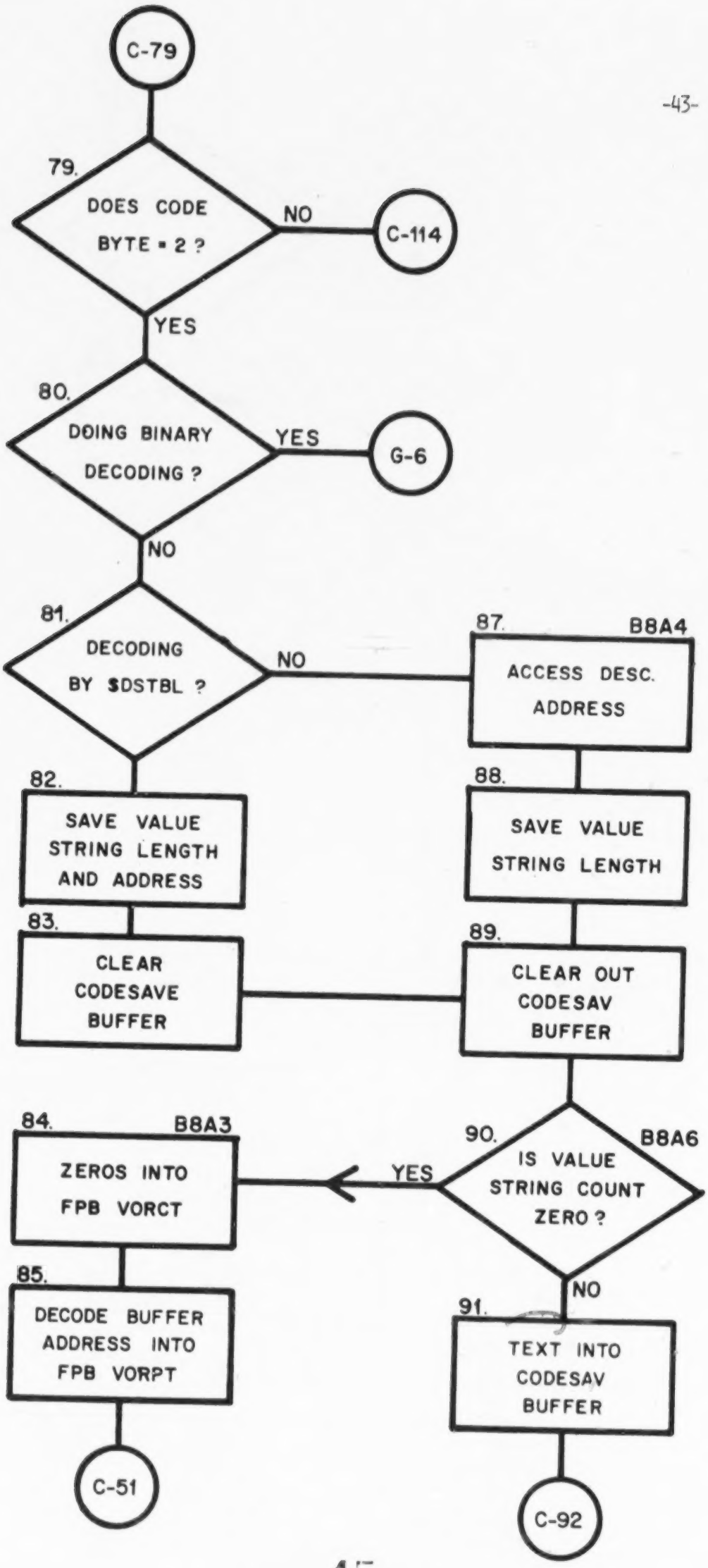

C-19
19.
B3
FIRST DESC. REQUEST ?
YES
NO
26.
B4A
ACCESS START OF RECORD
20.
NEXT DESC. REQUEST ?
YES
NO
31.
B4C
ACCESS START OF RECORD
32.
STEP TO CURRENT DESC.
27.
ADD FIRST DESC. HALF-WORD TO START ADDRESS
21.
N 1
SEARCH FOR DESCRIPTOR
33.
END OF RECORD ?
YES
C-114
NO
C-118
22.
IS DESC. IN RECORD ?
NO
34.
ADD CURRENT DESC. RAD TO START ADDRESS
28.
B4B
STORE RESULT AS FIRST DESC. ADDRESS
29.
MOVE DESC. NAME TO CALLER'S FPB
YES
23.
B3A
FETCH WITHOUT DECODE ?
NO
24.
IS $DSTBL MISSING ?
NO
C-36
YES
YES
C-41
C-41

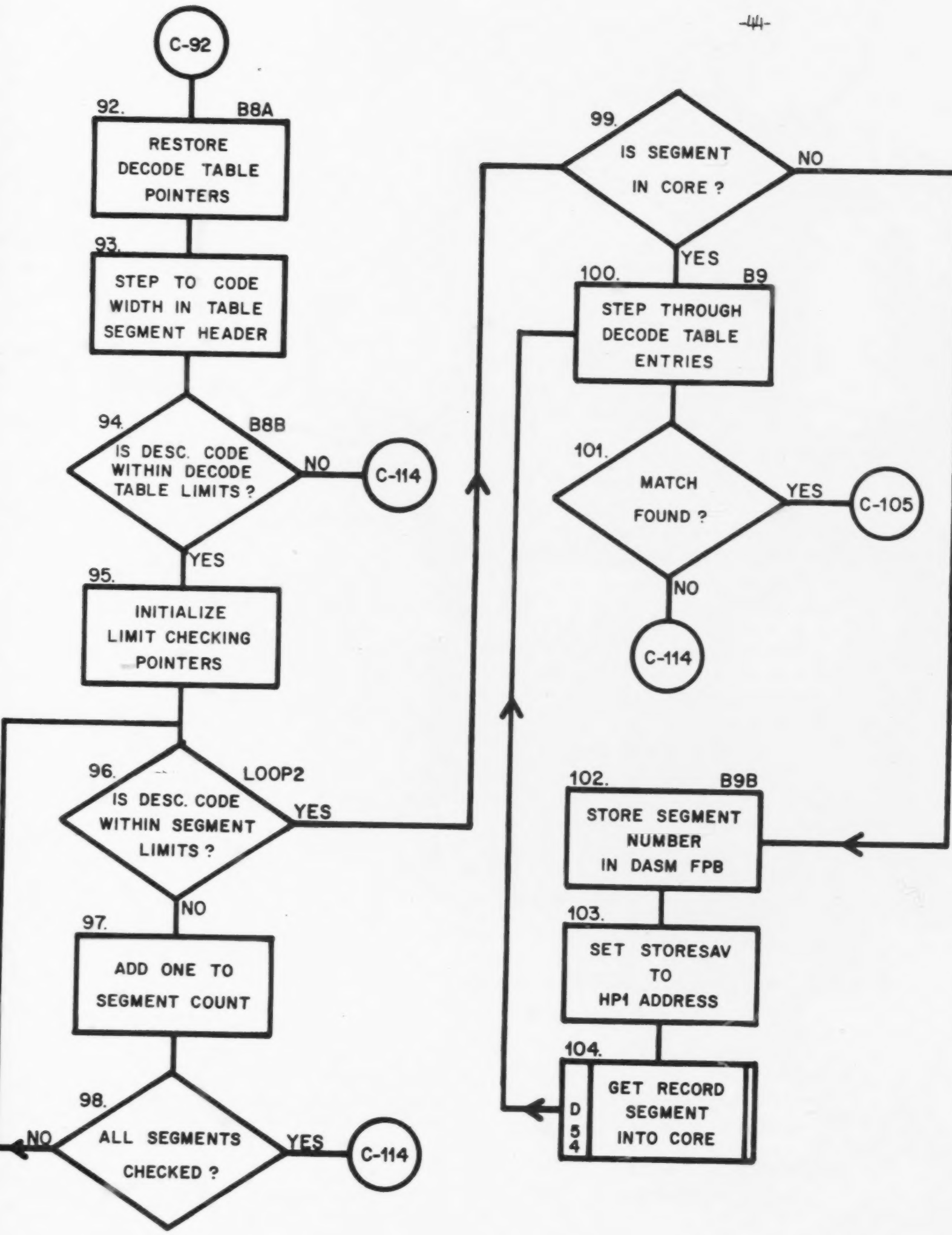

C-36
36.
B5
ACCESS
START OF HP6
37.
STEP TO
SECOND ENTRY
38.
IS THIS A
$DSTBL ?
NO
C-114
YES
39.
B5A
SEARCH FOR
DESC. NAME
40.
IS DESC.
ENTRY THERE
YES
C-59
NO
C-54

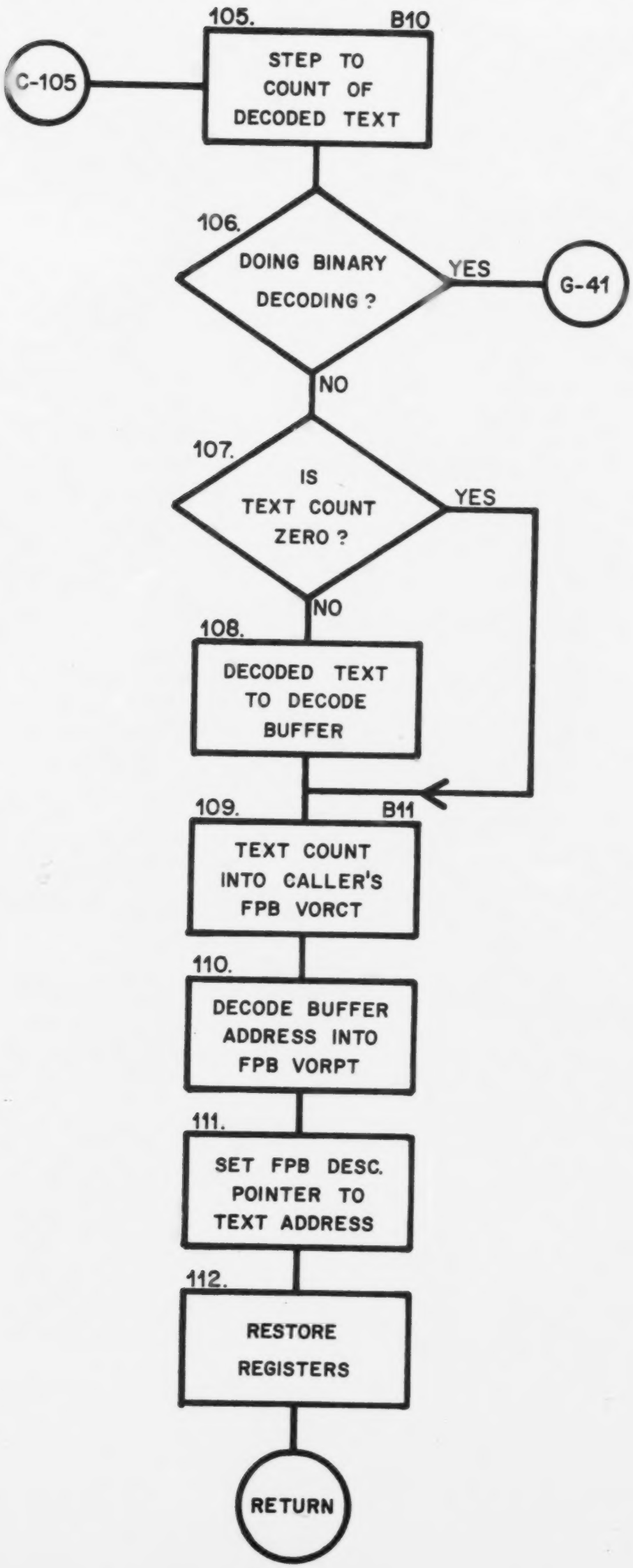

C-41
41.
B6
DOING BINARY DECODING ?
YES
42.
TURN OFF BINARY DECODE FLAG
C-114
NO
44.
DECODING BY DSTBL ?
NO
47.
ZERO OUT DASM FPB ERROR INDICATORS
YES
45.
SET FPB ERROR: NO DSTBL ENTRY FOR THIS DESC.
C-118
48.
STORE DESC. ADDRESS IN CALLER'S FPB
49.
STORE VALUE STRING COUNT IN FPB VORCT
50.
STORE VALUE STRING ADDRESS IN FPB VORPT
C-51
51.
RETURN
ZERO OUT FPB ERROR INDICATOR
52.
RESTORE ALL REGISTERS
RETURN

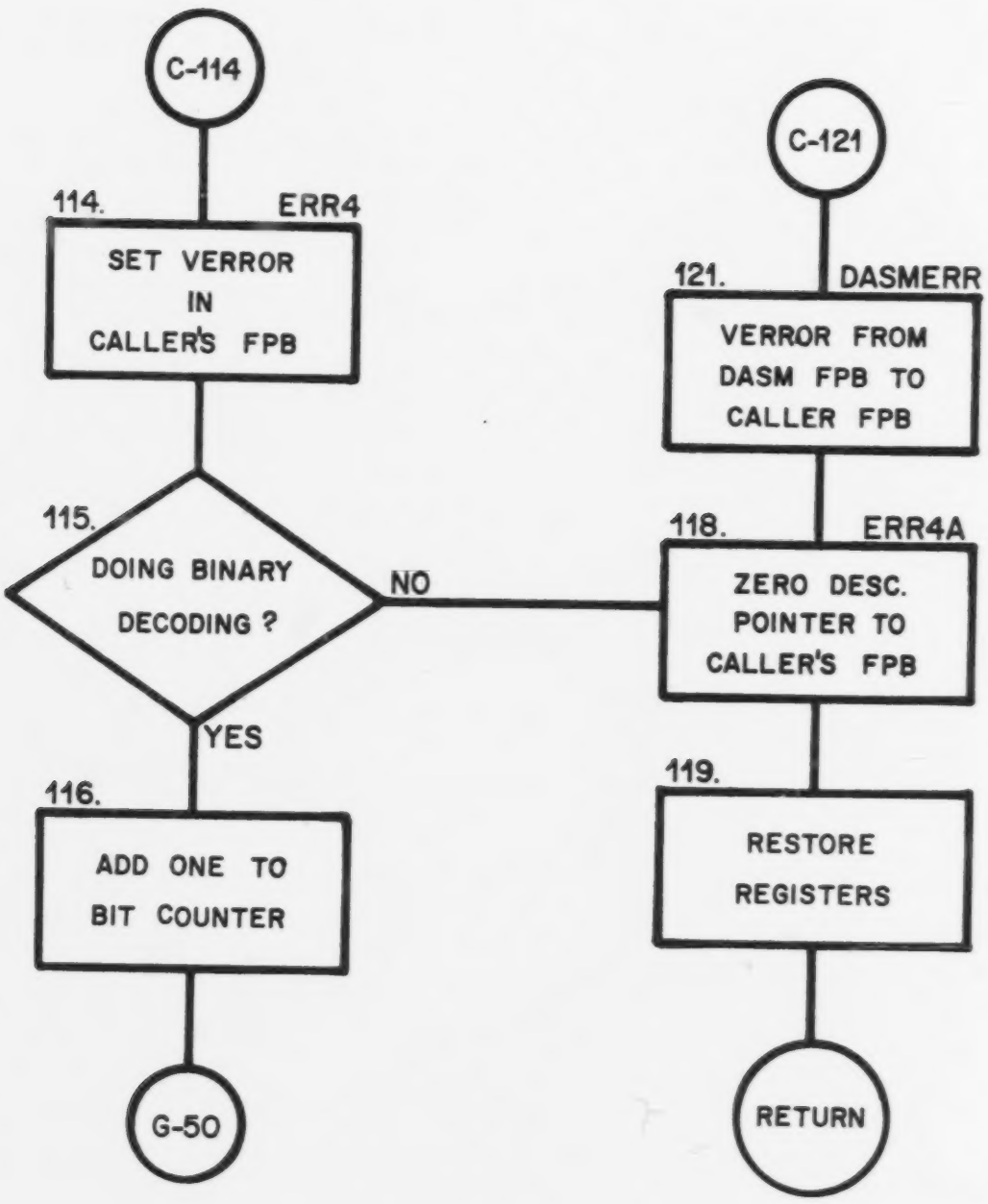

C-59
59.
B7
DECODE
BY $DSTBL ?
YES
NO
63.
B7B
BINARY
ENTRY ?
YES
G-1
NO
60.
DOING BINARY
DECODING ?
NO
YES
54.
B6A
SET STORESAV
TO
HP6 ADDRESS
C-54
61.
B7A
IS TABLE NAME
COUNT ZERO?
YES
C-41
NO
64.
B8
ACCESS
DASM FPB
55.
LAST
$DSTBL SEGMENT
IN CORE ?
YES
NO
56.
ACCESS
DASM FPB
65.
NO
SAME NAME AS
LAST FETCH ?
57.
D
5
4
NEXT RECORD
SEGMENT
INTO CORE
66.
PART OF
DECODE TABLE
IN CORE ?
YES
C-76
58.
WAS FETCH
SUCCESSFUL ?
NO
NO
67.
B8A1
CLEAR OUT
DASM FPB
C-68
YES
C-36
C-41

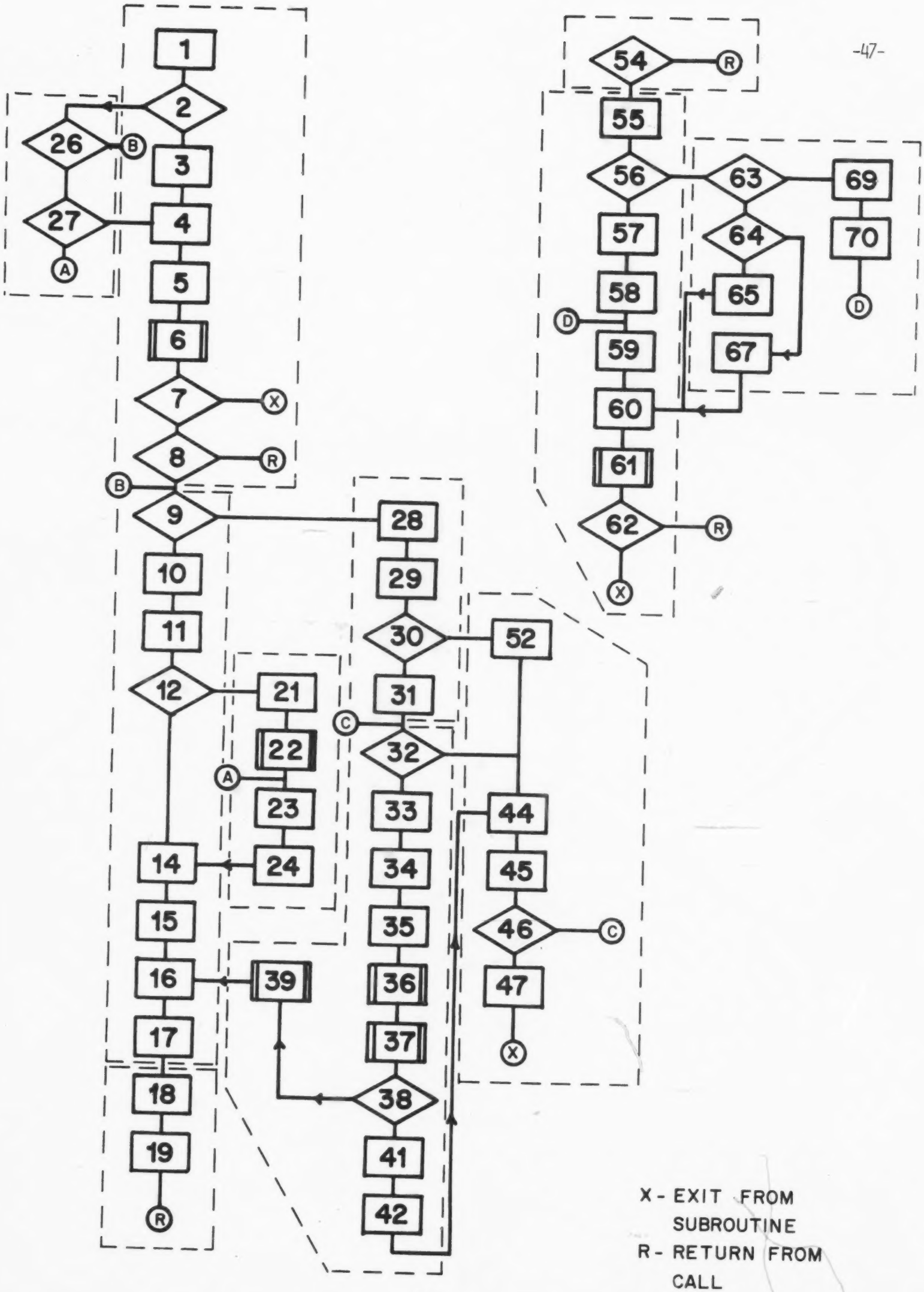

C-68
68.
DECODE TABLE NAME TO DASM FPB
69.
MAKE RECORD NAME COUNT EVEN
72.
B8A2
SET SEQUENCE COUNT TO ZERO
73.
SET STORESAV TO HP1 ADDRESS
74.
M 1
FETCH RECORD INTO CORE
75.
WAS FETCH SUCCESSFUL ?
NO
C-121
YES
C-76
76.
B8A
ACCESS START OF DECODE TABLE RECORD
77.
STEP TO START OF TABLE SEGMENT HEADER
78.
SAVE TABLE SEGMENT HEADER IN TABSAVE
C-79


C-79
79.
DOES CODE BYTE = 2 ?
NO
C-114
YES
80.
DOING BINARY DECODING ?
YES
G-6
NO
81.
DECODING BY $DSTBL ?
NO
87.
B8A4
ACCESS DESC. ADDRESS
82.
SAVE VALUE STRING LENGTH AND ADDRESS
88.
SAVE VALUE STRING LENGTH
83.
CLEAR CODESAVE BUFFER
89.
CLEAR OUT CODESAV BUFFER
84.
B8A3
ZEROS INTO FPB VORCT
YES
90.
B8A6
IS VALUE STRING COUNT ZERO ?
85.
DECODE BUFFER ADDRESS INTO FPB VORPT
NO
91.
TEXT INTO CODESAV BUFFER
C-51
C-92

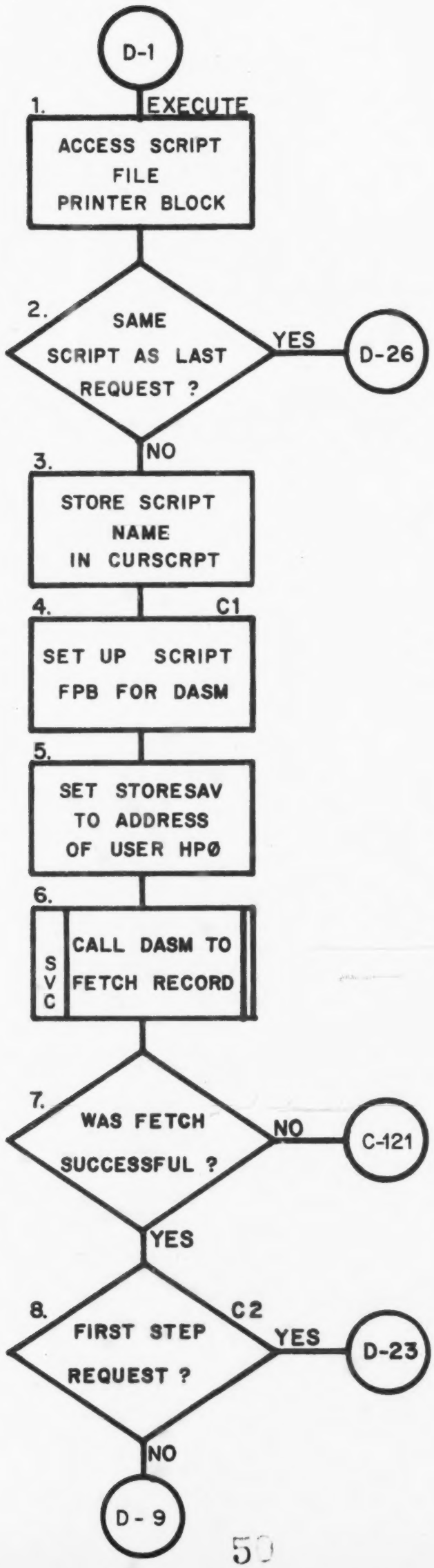

C-92
92.
B8A
RESTORE DECODE TABLE POINTERS
93.
STEP TO CODE WIDTH IN TABLE SEGMENT HEADER
94.
B8B
IS DESC. CODE WITHIN DECODE TABLE LIMITS ?
NO
C-114
YES
95.
INITIALIZE LIMIT CHECKING POINTERS
96.
LOOP2
IS DESC. CODE WITHIN SEGMENT LIMITS ?
YES
NO
97.
ADD ONE TO SEGMENT COUNT
98.
ALL SEGMENTS CHECKED ?
NO
YES
C-114
99.
IS SEGMENT IN CORE ?
NO
YES
100.
B9
STEP THROUGH DECODE TABLE ENTRIES
101.
MATCH FOUND ?
YES
C-105
NO
C-114
102.
B9B
STORE SEGMENT NUMBER IN DASM FPB
103.
SET STORESAV TO HP1 ADDRESS
104.
D 5 4
GET RECORD SEGMENT INTO CORE

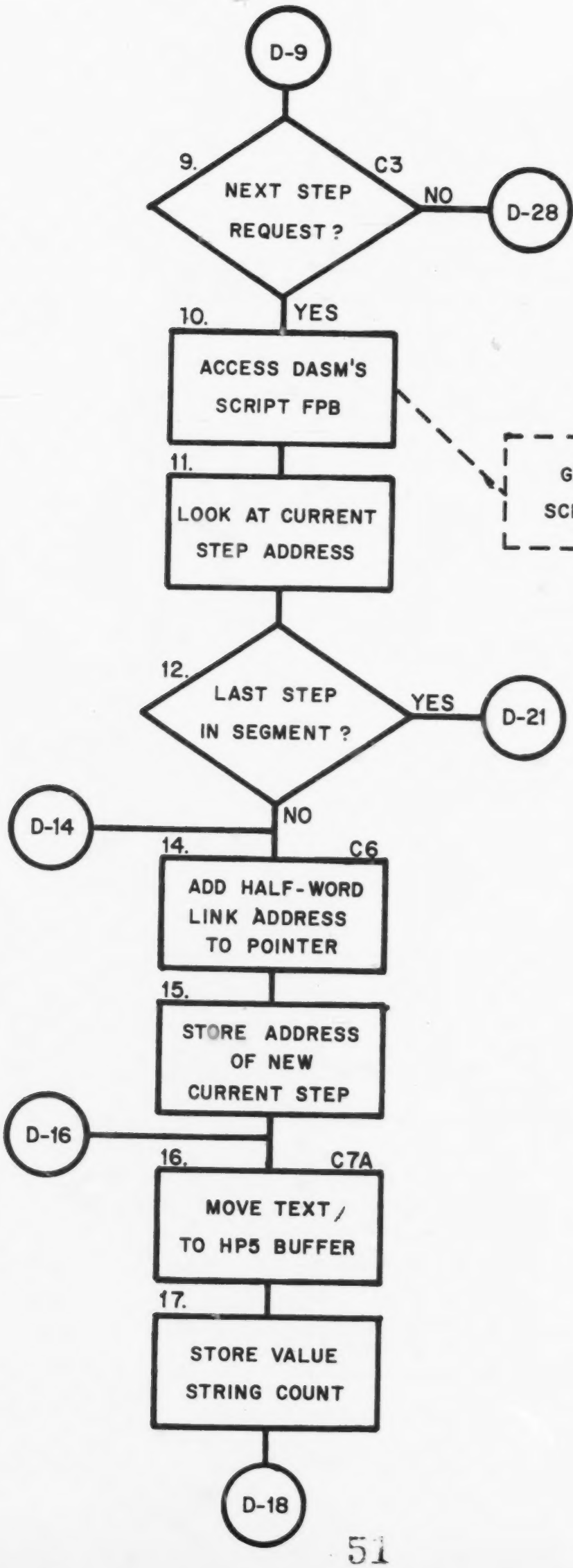

105.
B10
STEP TO COUNT OF DECODED TEXT
C-105
106.
DOING BINARY DECODING ?
YES
G-41
NO
107.
IS TEXT COUNT ZERO ?
YES
NO
108.
DECODED TEXT TO DECODE BUFFER
109.
B11
TEXT COUNT INTO CALLER'S FPB VORCT
110.
DECODE BUFFER ADDRESS INTO FPB VORPT
111.
SET FPB DESC. POINTER TO TEXT ADDRESS
112.
RESTORE REGISTERS
RETURN

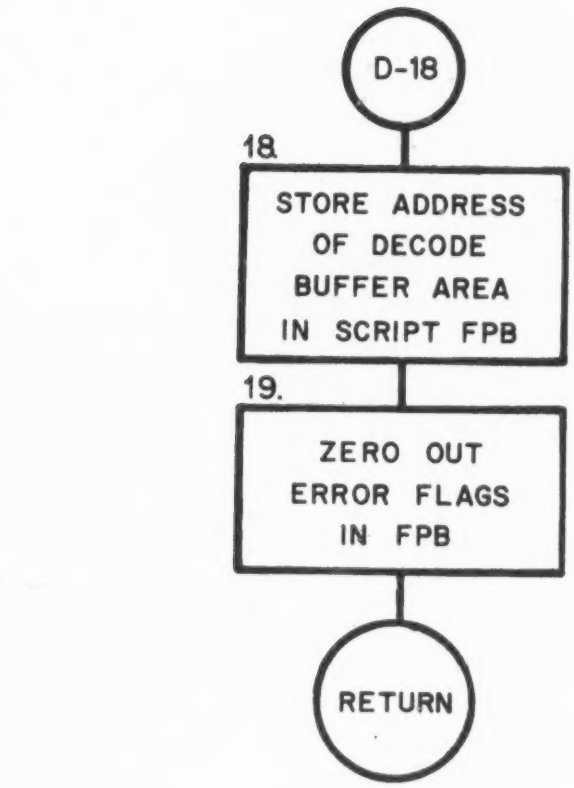

C-114
114.
ERR4
SET VERROR IN CALLER'S FPB
115.
DOING BINARY DECODING ?
NO
YES
116.
ADD ONE TO BIT COUNTER
G-50
C-121
121.
DASMERR
VERROR FROM DASM FPB TO CALLER FPB
118.
ERR4A
ZERO DESC. POINTER TO CALLER'S FPB
119.
RESTORE REGISTERS
RETURN

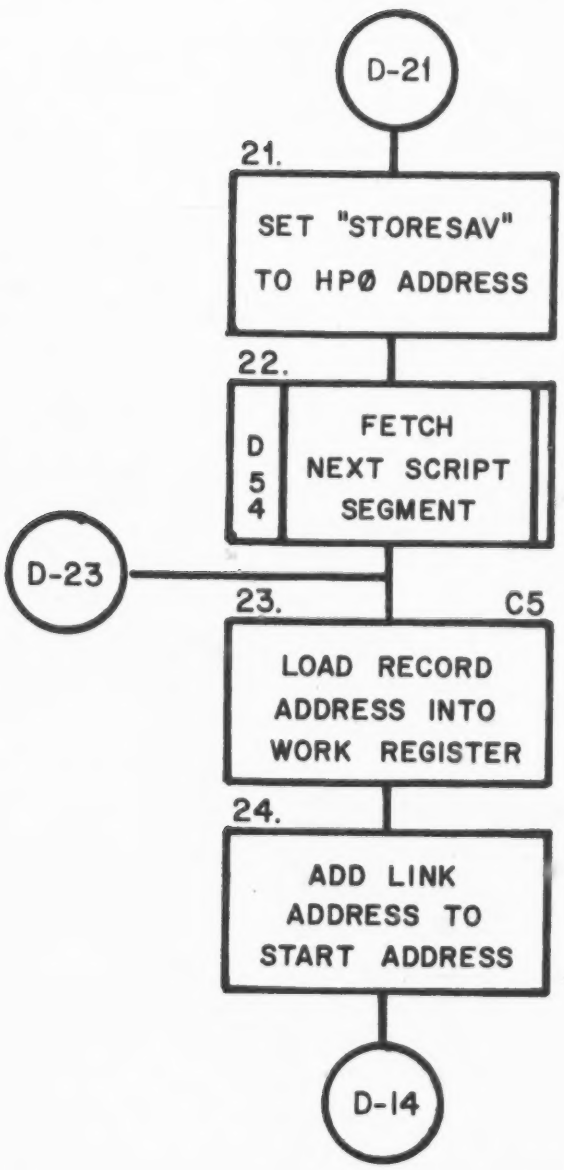


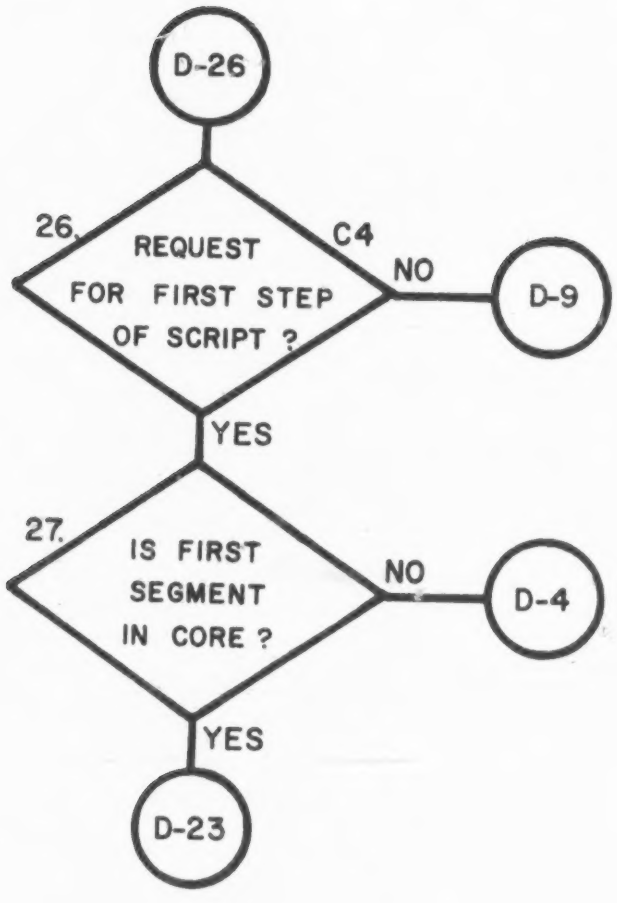
X - EXIT FROM SUBROUTINE
R - RETURN FROM CALL

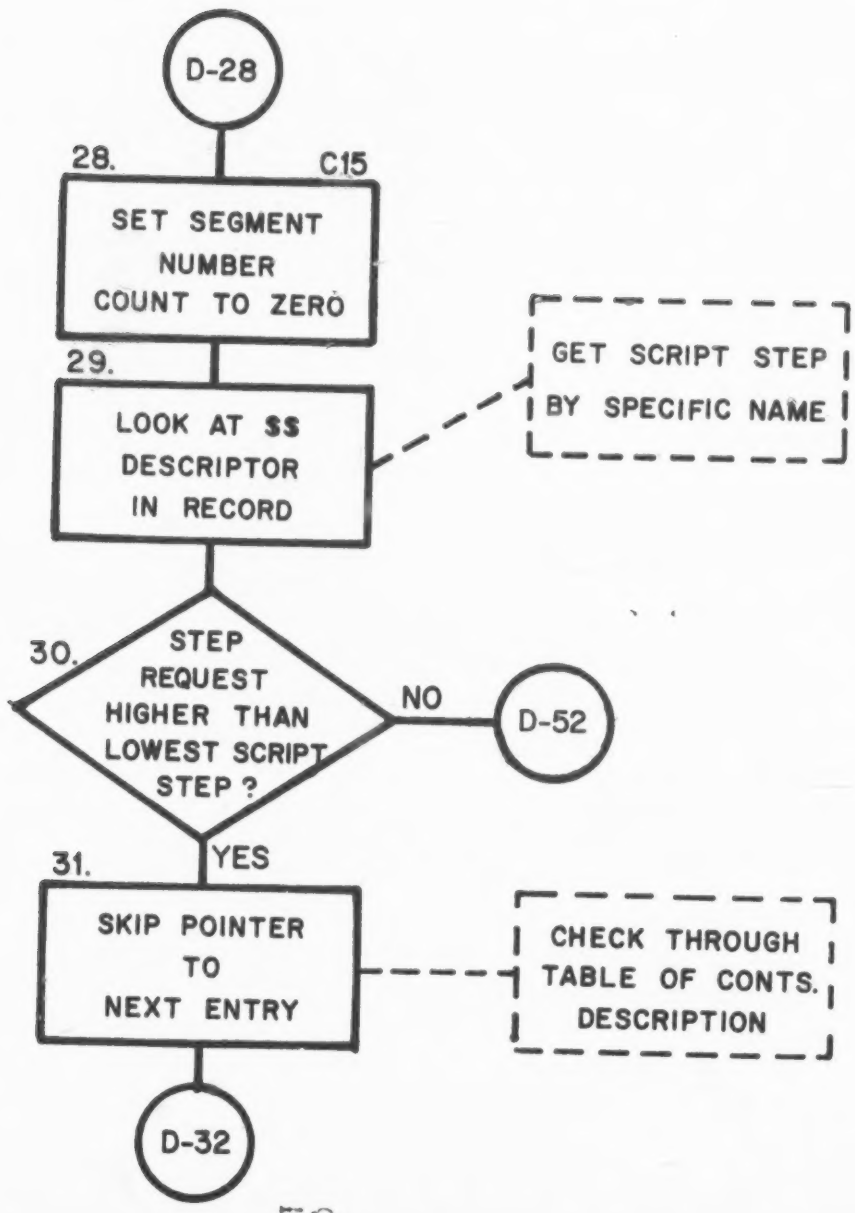

D-1
1.
EXECUTE
ACCESS SCRIPT FILE PRINTER BLOCK
2.
SAME SCRIPT AS LAST REQUEST ?
YES
D-26
NO
3.
STORE SCRIPT NAME IN CURSCRPT
4.
C1
SET UP SCRIPT FPB FOR DASM
5.
SET STORESAV TO ADDRESS OF USER HPØ
6.
SVC
CALL DASM TO FETCH RECORD
7.
WAS FETCH SUCCESSFUL ?
NO
C-121
YES
8.
C2
FIRST STEP REQUEST ?
YES
D-23
NO
D - 9

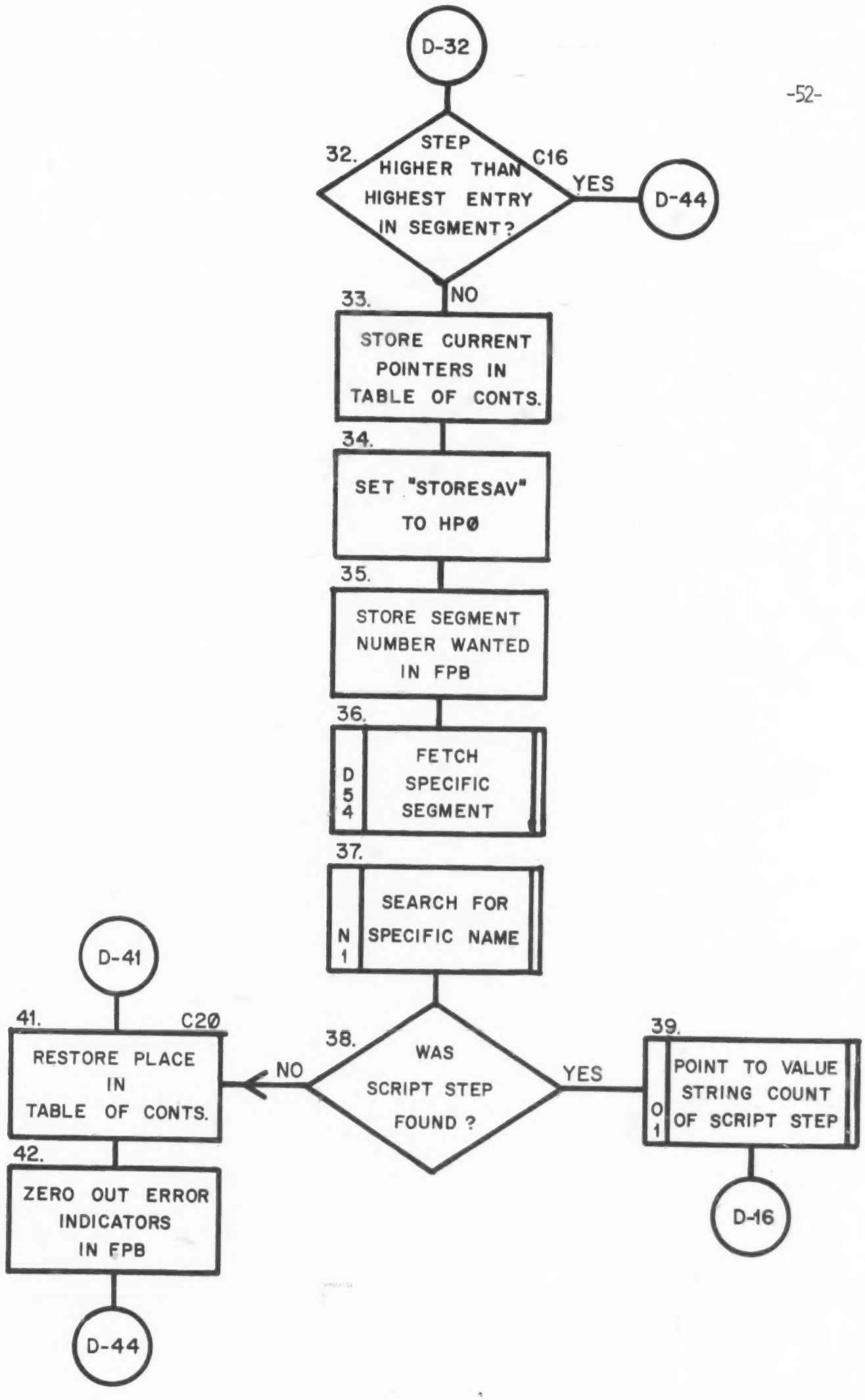

D-9
9.
C3
NEXT STEP
REQUEST ?
NO
D-28
YES
10.
ACCESS DASM'S
SCRIPT FPB
GET NEXT
SCRIPT STEP
11.
LOOK AT CURRENT
STEP ADDRESS
12.
LAST STEP
IN SEGMENT ?
YES
D-21
NO
D-14
14.
C6
ADD HALF-WORD
LINK ADDRESS
TO POINTER
15.
STORE ADDRESS
OF NEW
CURRENT STEP
D-16
16.
C7A
MOVE TEXT
TO HP5 BUFFER
17.
STORE VALUE
STRING COUNT
D-18

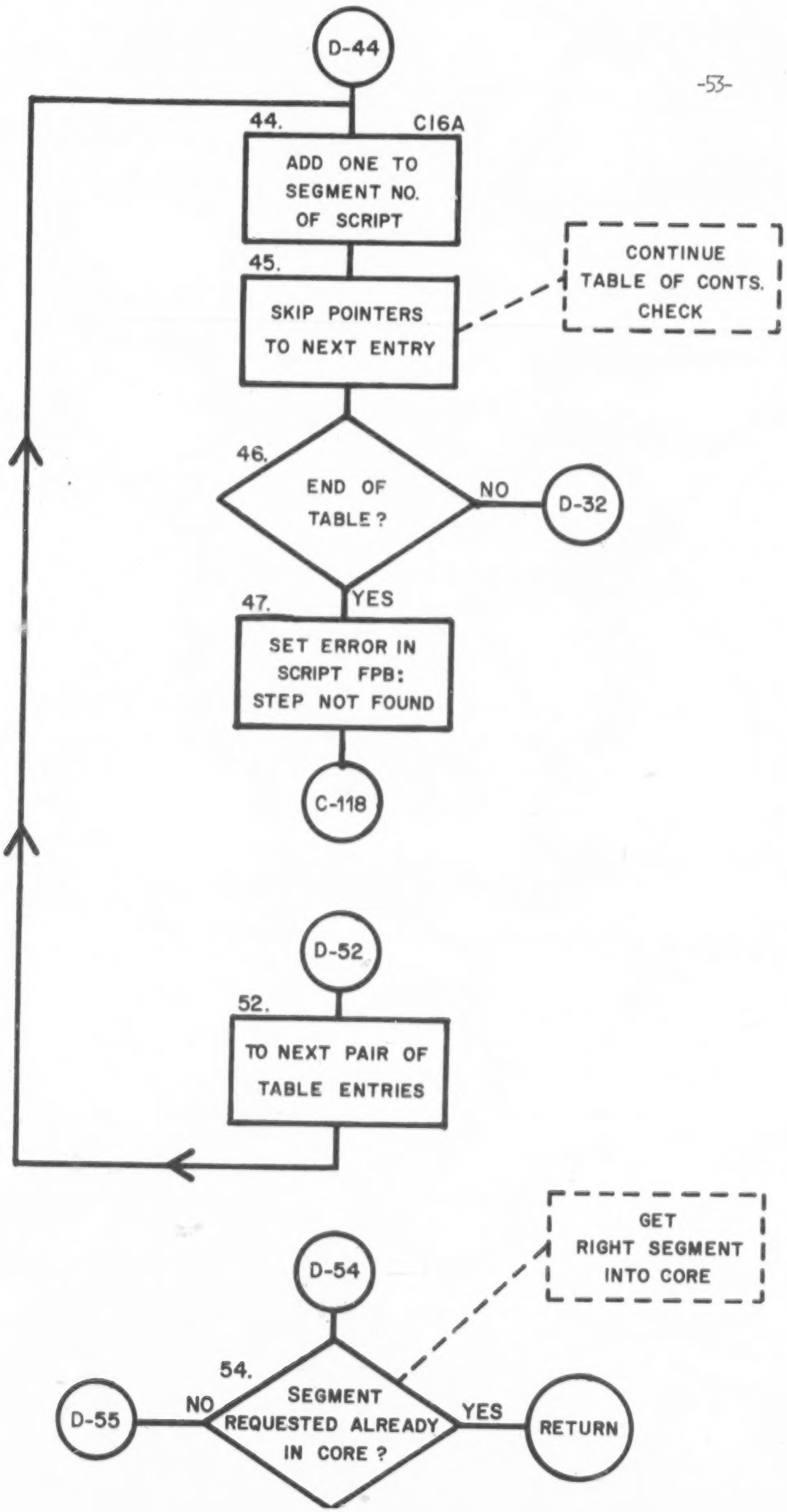

D-18
18.
STORE ADDRESS OF DECODE BUFFER AREA IN SCRIPT FPB
19.
ZERO OUT ERROR FLAGS IN FPB
RETURN

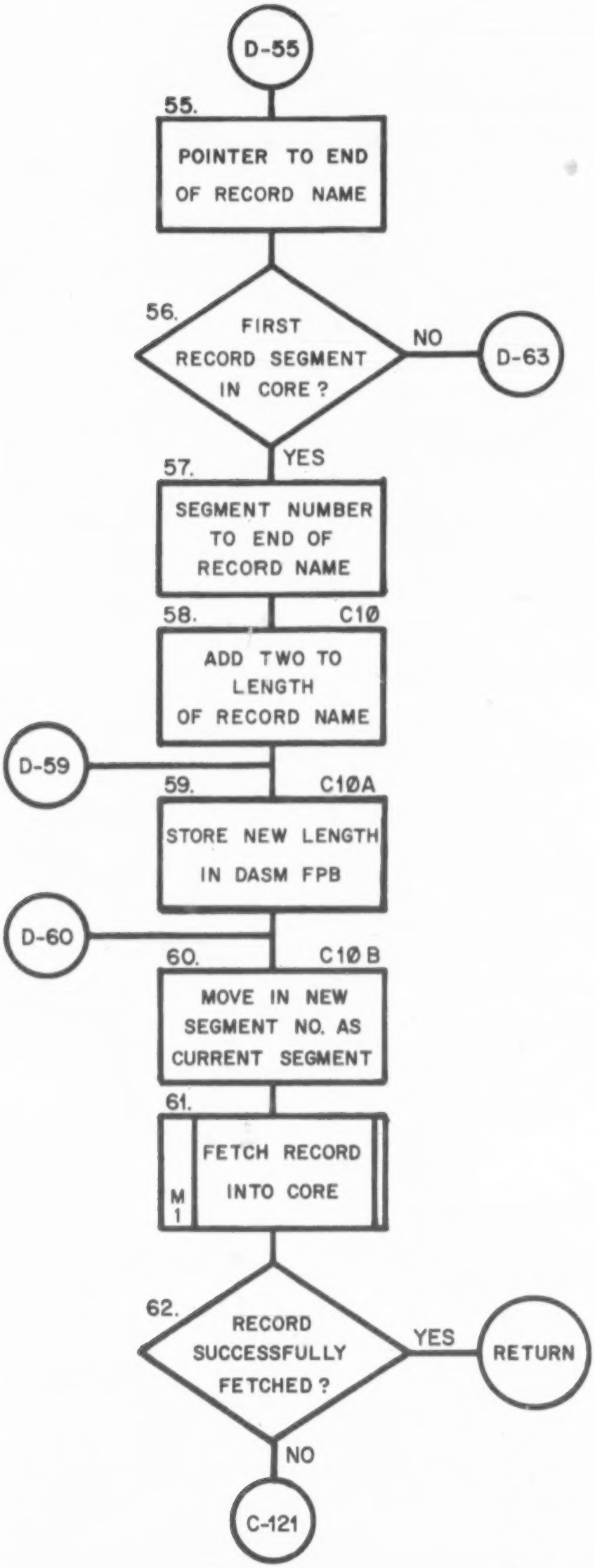

D-21
21.
SET "STORESAV" TO HPØ ADDRESS
22.
D 5 4
FETCH NEXT SCRIPT SEGMENT
D-23
23.
C5
LOAD RECORD ADDRESS INTO WORK REGISTER
24.
ADD LINK ADDRESS TO START ADDRESS
D-14

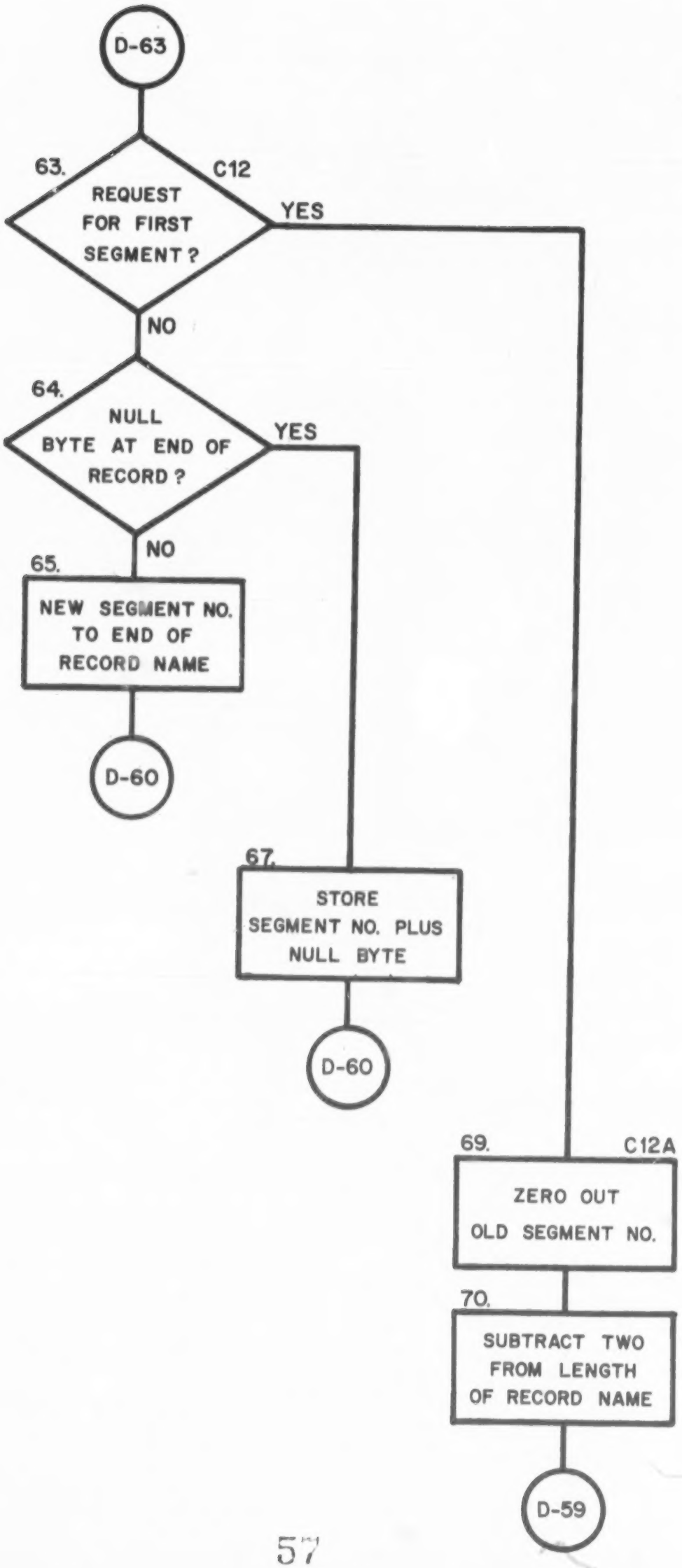

D-26
26.
REQUEST
FOR FIRST STEP
OF SCRIPT ?
C4
NO
D-9
YES
27.
IS FIRST
SEGMENT
IN CORE ?
NO
D-4
YES
D-23

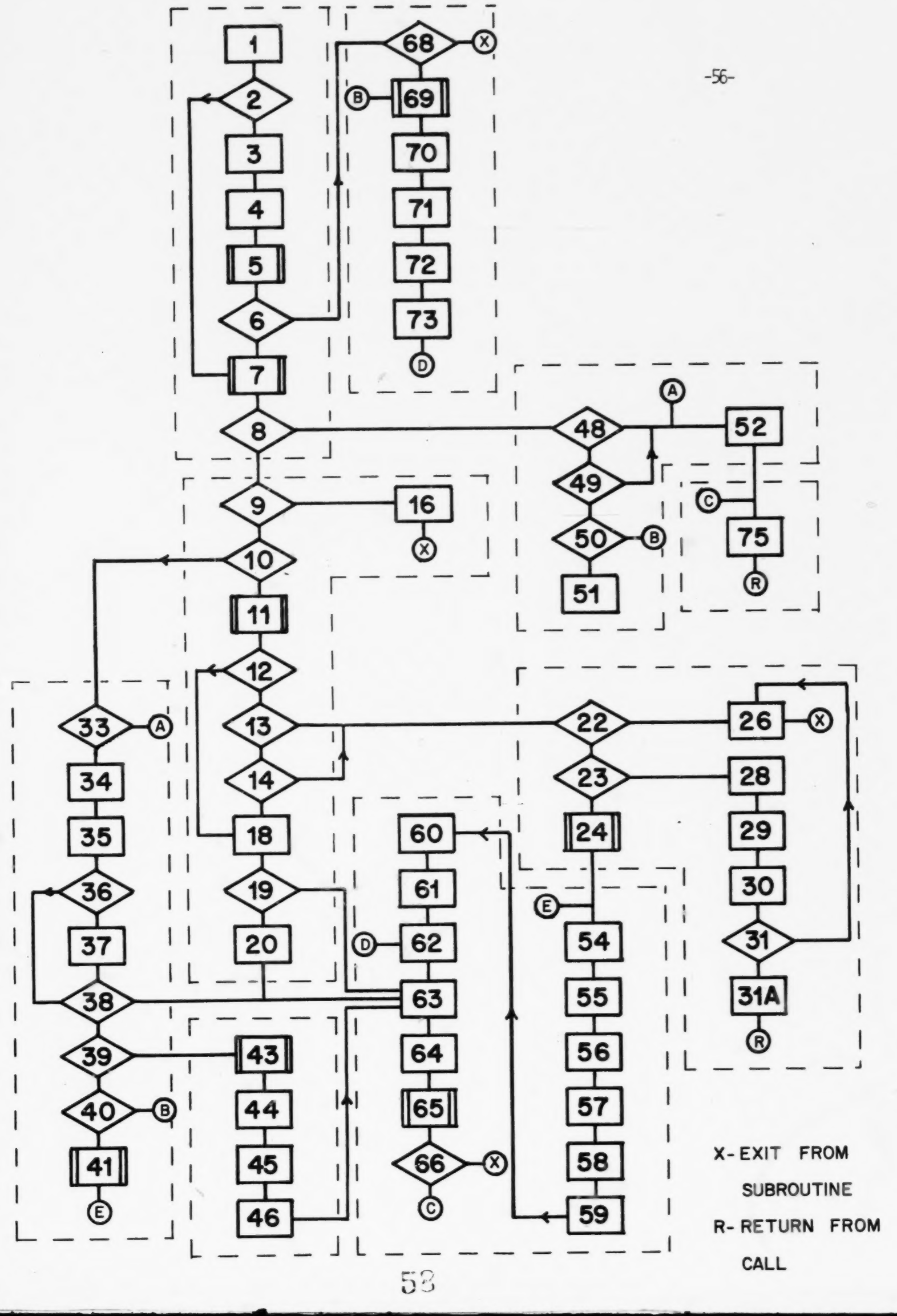

D-28
28.
SET SEGMENT
NUMBER
COUNT TO ZERO
C15
29.
LOOK AT $$
DESCRIPTOR
IN RECORD
GET SCRIPT STEP
BY SPECIFIC NAME
30.
STEP
REQUEST
HIGHER THAN
LOWEST SCRIPT
STEP ?
NO
D-52
YES
31.
SKIP POINTER
TO
NEXT ENTRY
CHECK THROUGH
TABLE OF CONTS.
DESCRIPTION
D-32

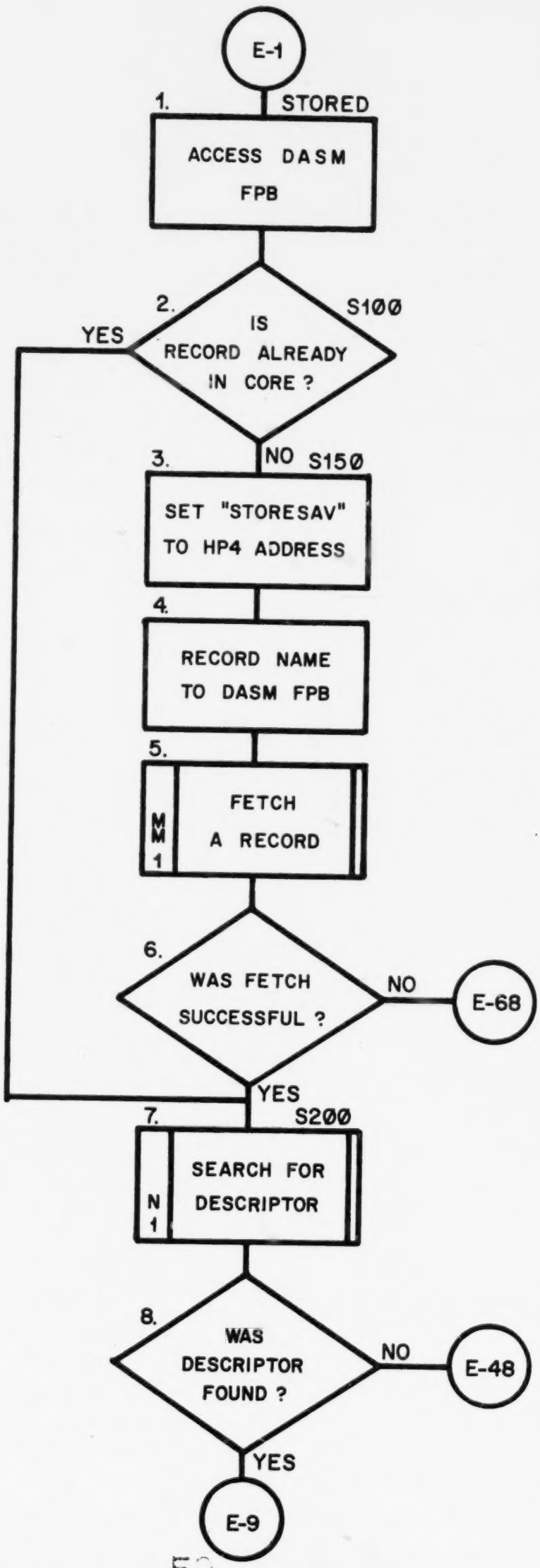

D-32
32.
STEP HIGHER THAN HIGHEST ENTRY IN SEGMENT?
C16
YES
D-44
NO
33.
STORE CURRENT POINTERS IN TABLE OF CONTS.
34.
SET "STORESAV" TO HPØ
35.
STORE SEGMENT NUMBER WANTED IN FPB
36.
D 5 4
FETCH SPECIFIC SEGMENT
37.
N 1
SEARCH FOR SPECIFIC NAME
D-41
41.
C2Ø
RESTORE PLACE IN TABLE OF CONTS.
38.
WAS SCRIPT STEP FOUND ?
NO
YES
39.
O 1
POINT TO VALUE STRING COUNT OF SCRIPT STEP
42.
ZERO OUT ERROR INDICATORS IN FPB
D-44
D-16

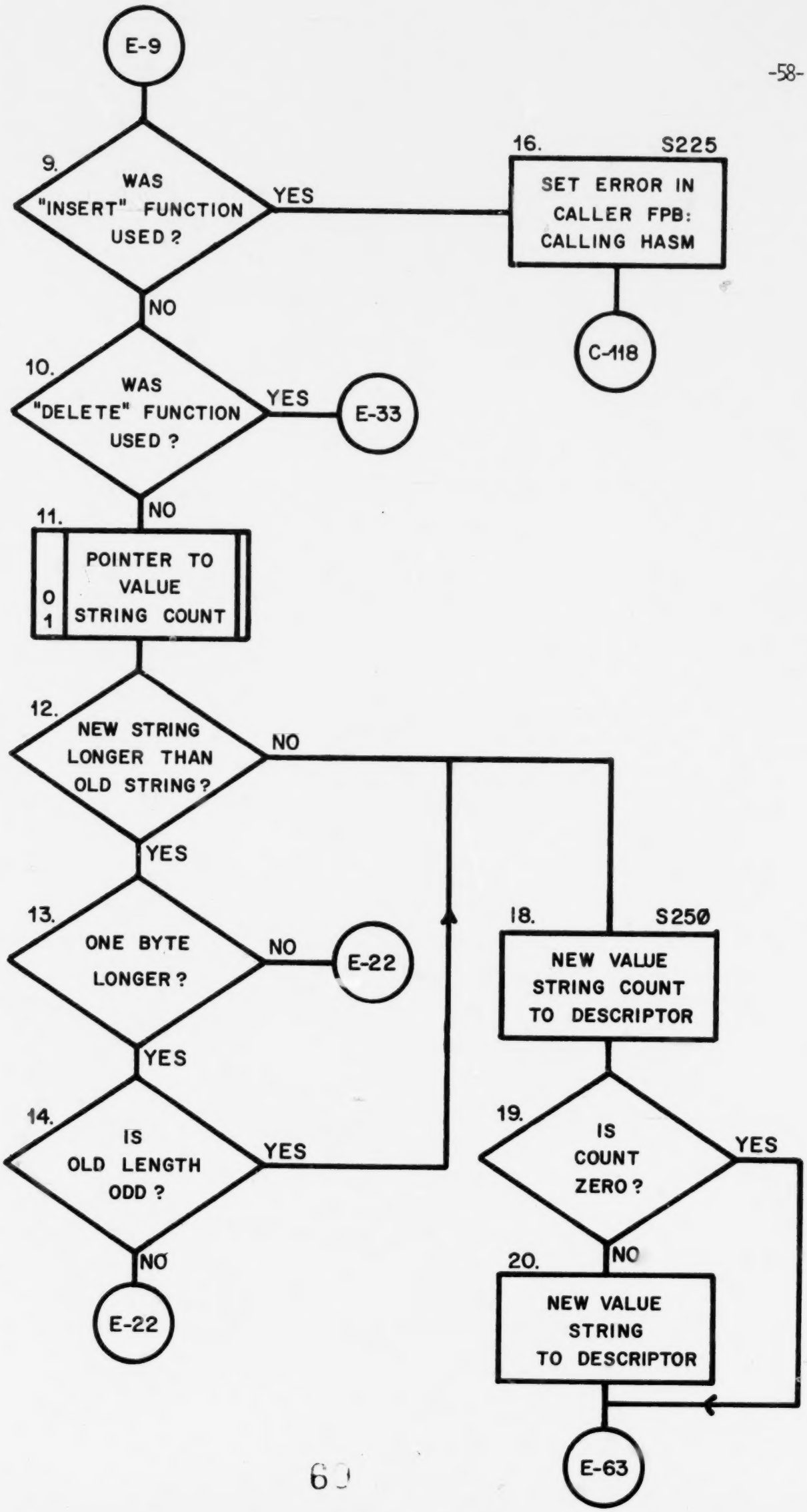

D-44
44.
C16A
ADD ONE TO SEGMENT NO. OF SCRIPT
45.
SKIP POINTERS TO NEXT ENTRY
CONTINUE TABLE OF CONTS. CHECK
46.
END OF TABLE?
NO
D-32
YES
47.
SET ERROR IN SCRIPT FPB: STEP NOT FOUND
C-118
D-52
52.
TO NEXT PAIR OF TABLE ENTRIES
GET RIGHT SEGMENT INTO CORE
D-54
54.
SEGMENT REQUESTED ALREADY IN CORE ?
NO
D-55
YES
RETURN

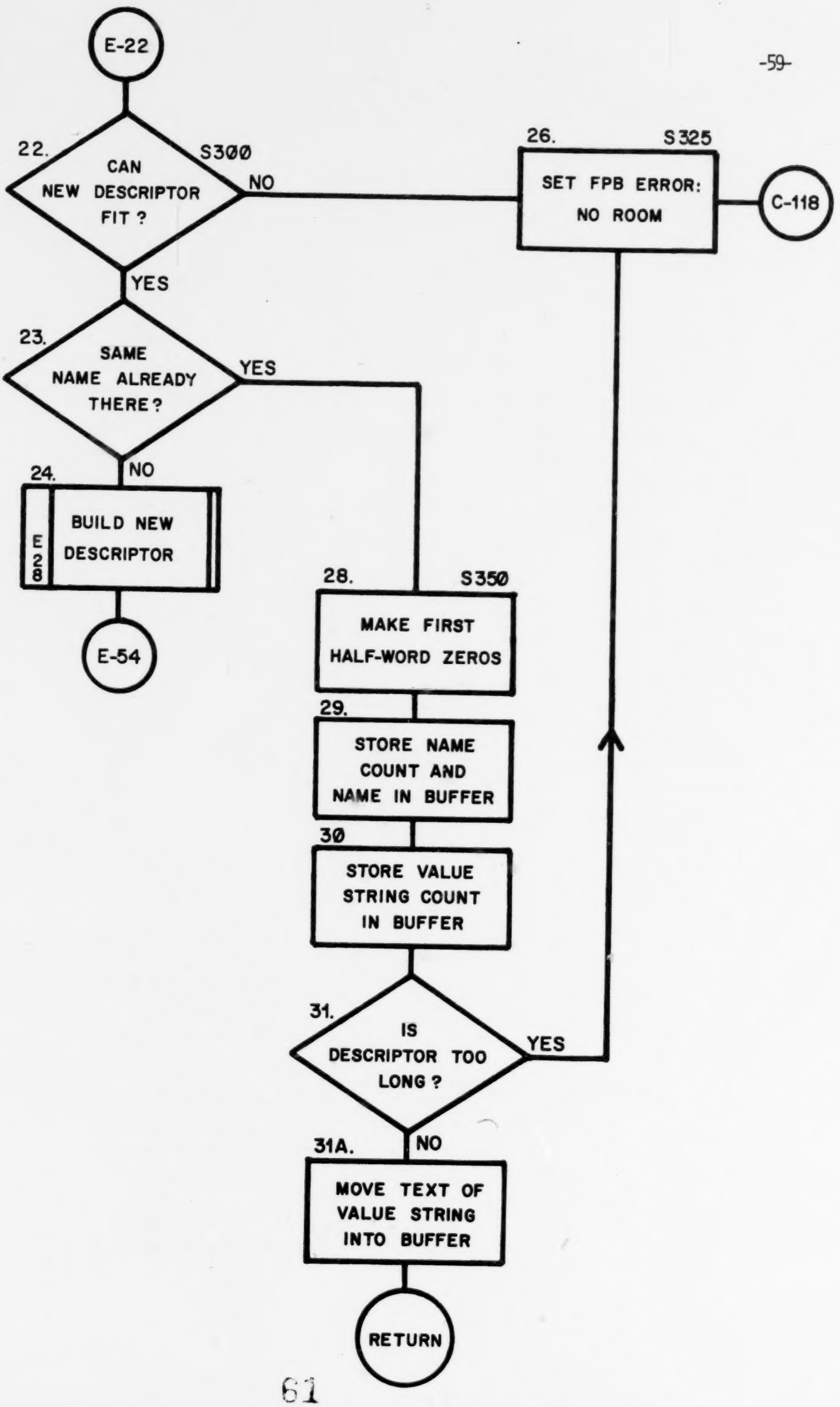

D-55
55.
POINTER TO END OF RECORD NAME
56.
FIRST RECORD SEGMENT IN CORE ?
NO
D-63
YES
57.
SEGMENT NUMBER TO END OF RECORD NAME
58.
C10
ADD TWO TO LENGTH OF RECORD NAME
D-59
59.
C10A
STORE NEW LENGTH IN DASM FPB
D-60
60.
C10B
MOVE IN NEW SEGMENT NO. AS CURRENT SEGMENT
61.
M 1
FETCH RECORD INTO CORE
62.
RECORD SUCCESSFULLY FETCHED ?
YES
RETURN
NO
C-121

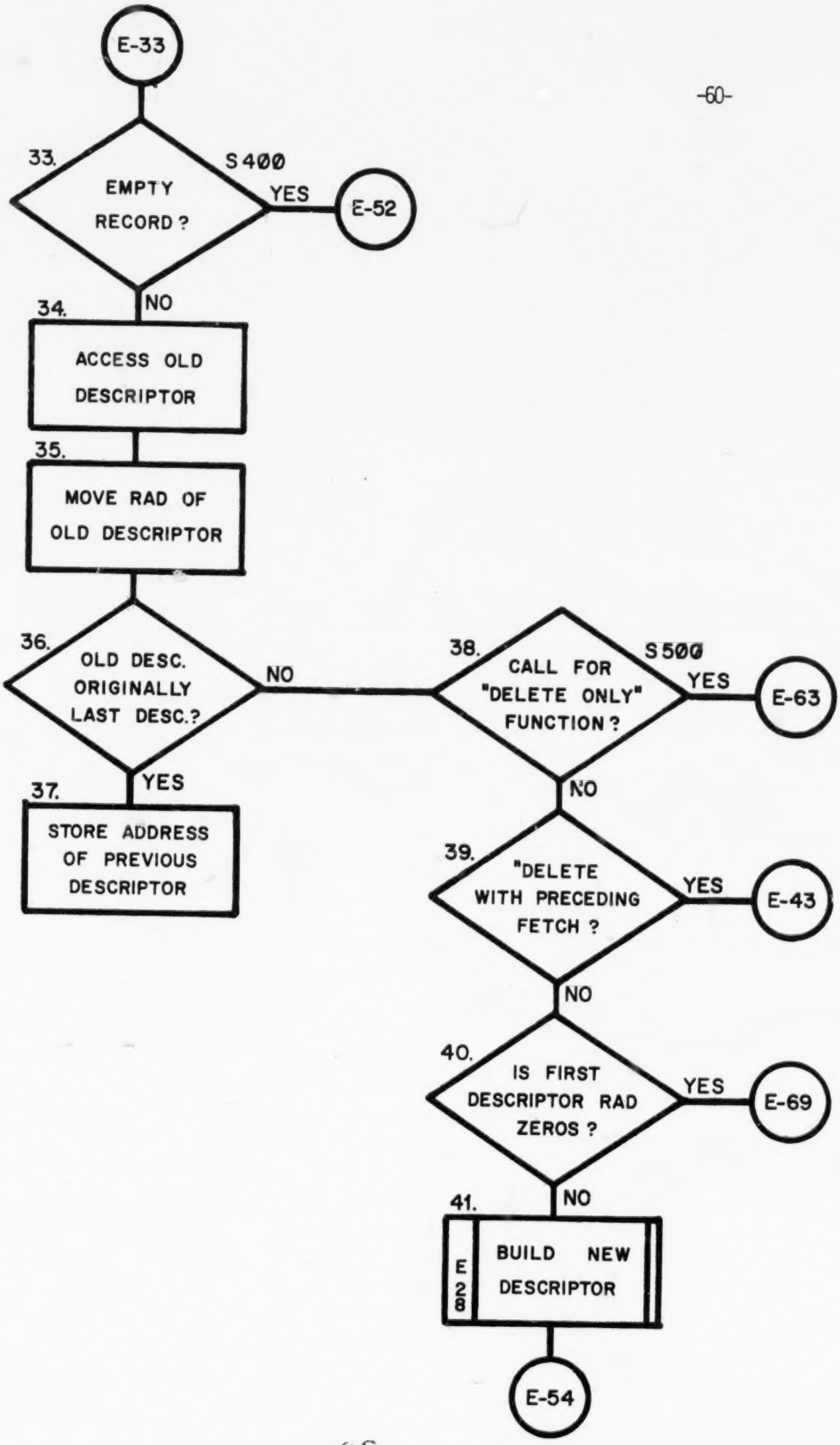

D-63
63.
C12
REQUEST FOR FIRST SEGMENT ?
YES
NO
64.
NULL BYTE AT END OF RECORD ?
YES
NO
65.
NEW SEGMENT NO. TO END OF RECORD NAME
D-60
67.
STORE SEGMENT NO. PLUS NULL BYTE
D-60
69.
C12A
ZERO OUT OLD SEGMENT NO.
70.
SUBTRACT TWO FROM LENGTH OF RECORD NAME
D-59

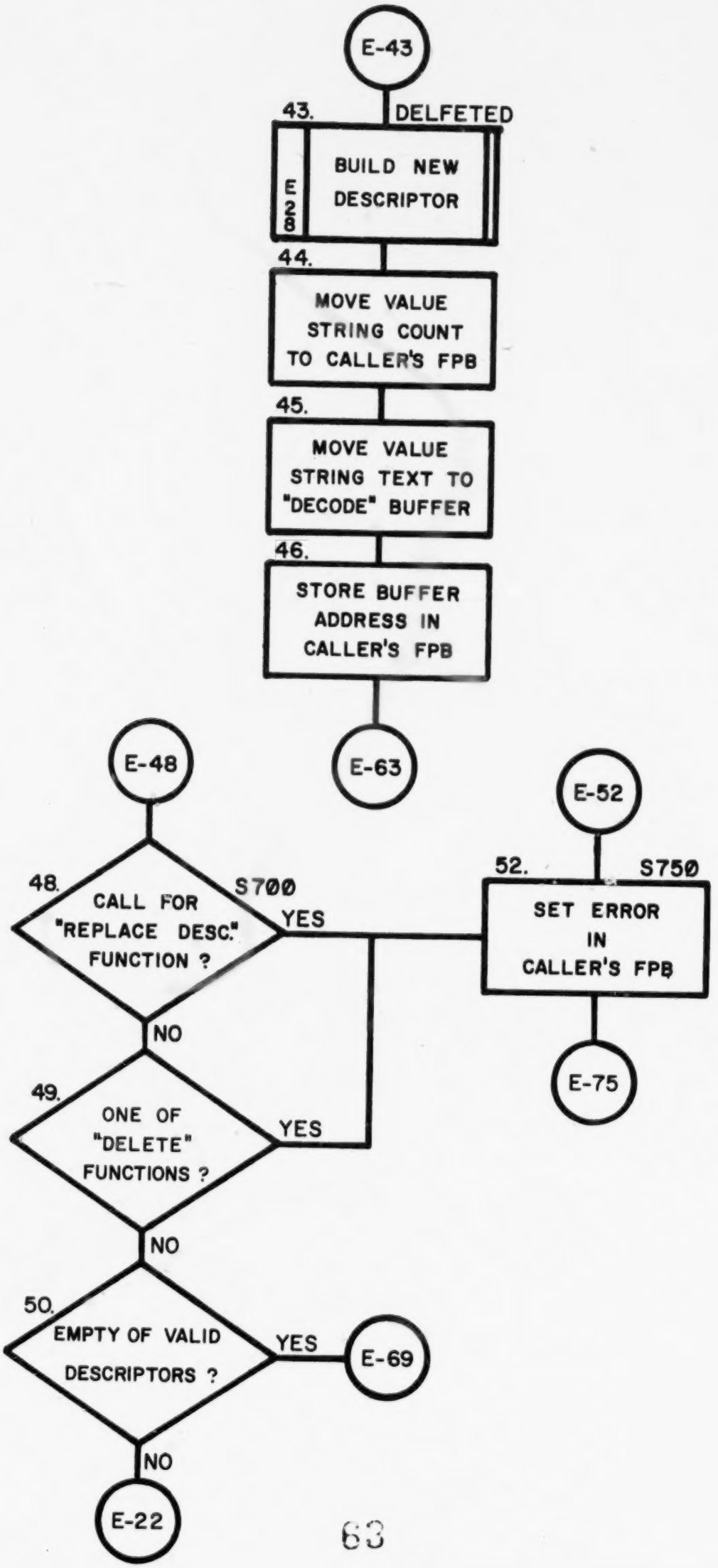

1
2
3
4
5
6
7
8
9
10
11
12
13
14
16
18
19
20
33
34
35
36
37
38
39
40
41
43
44
45
46
60
61
62
63
64
65
66
68
69
70
71
72
73
22
23
24
26
28
29
30
31
31A
48
49
50
51
52
54
55
56
57
58
59
75
A
B
C
D
E
R
X
X- EXIT FROM SUBROUTINE
R- RETURN FROM CALL

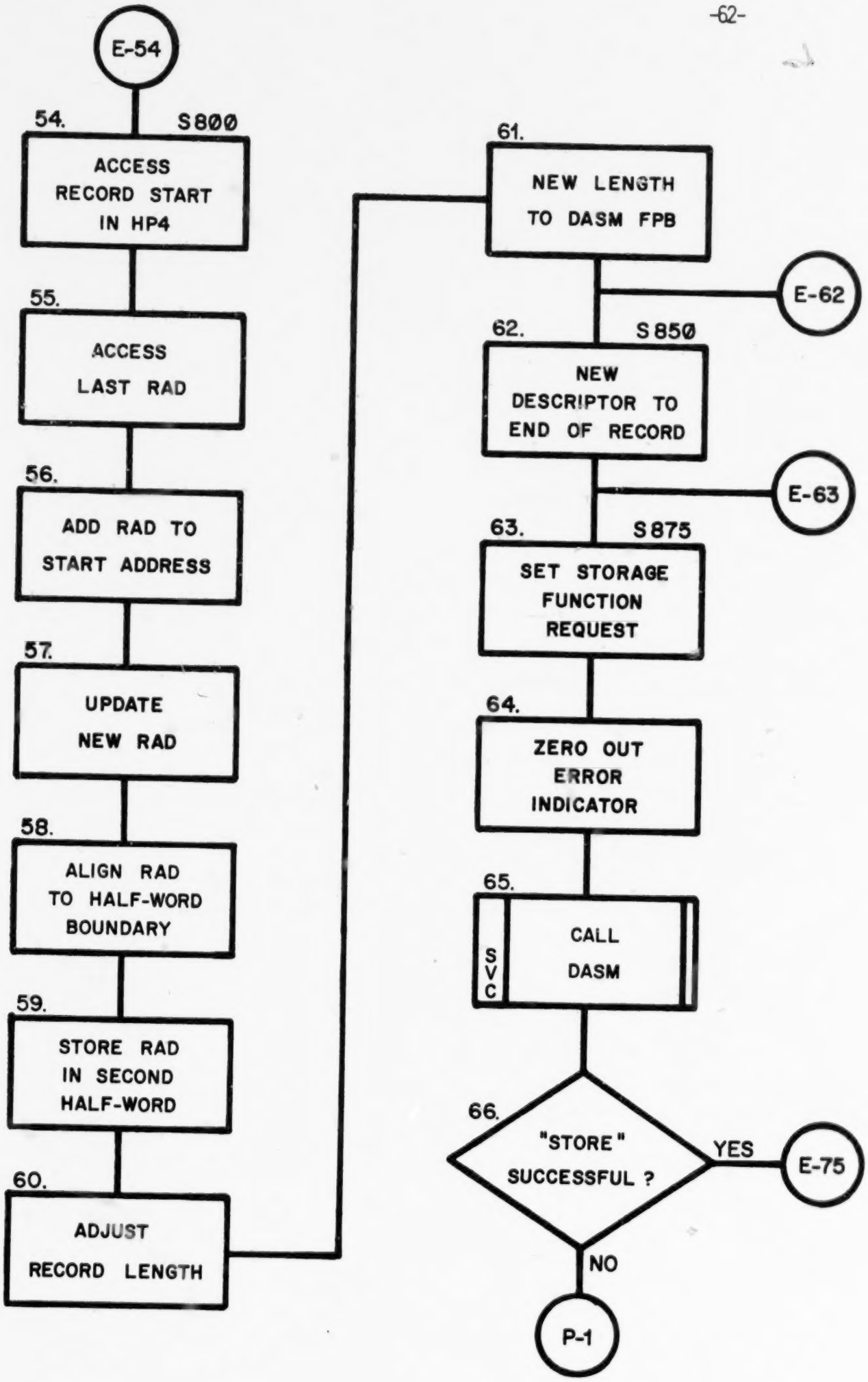

E-1
1.
STORED
ACCESS DASM FPB
2.
S100
IS RECORD ALREADY IN CORE ?
YES
NO
3.
S150
SET "STORESAV" TO HP4 ADDRESS
4.
RECORD NAME TO DASM FPB
5.
MM 1
FETCH A RECORD
6.
WAS FETCH SUCCESSFUL ?
NO
E-68
YES
7.
S200
N 1
SEARCH FOR DESCRIPTOR
8.
WAS DESCRIPTOR FOUND ?
NO
E-48
YES
E-9

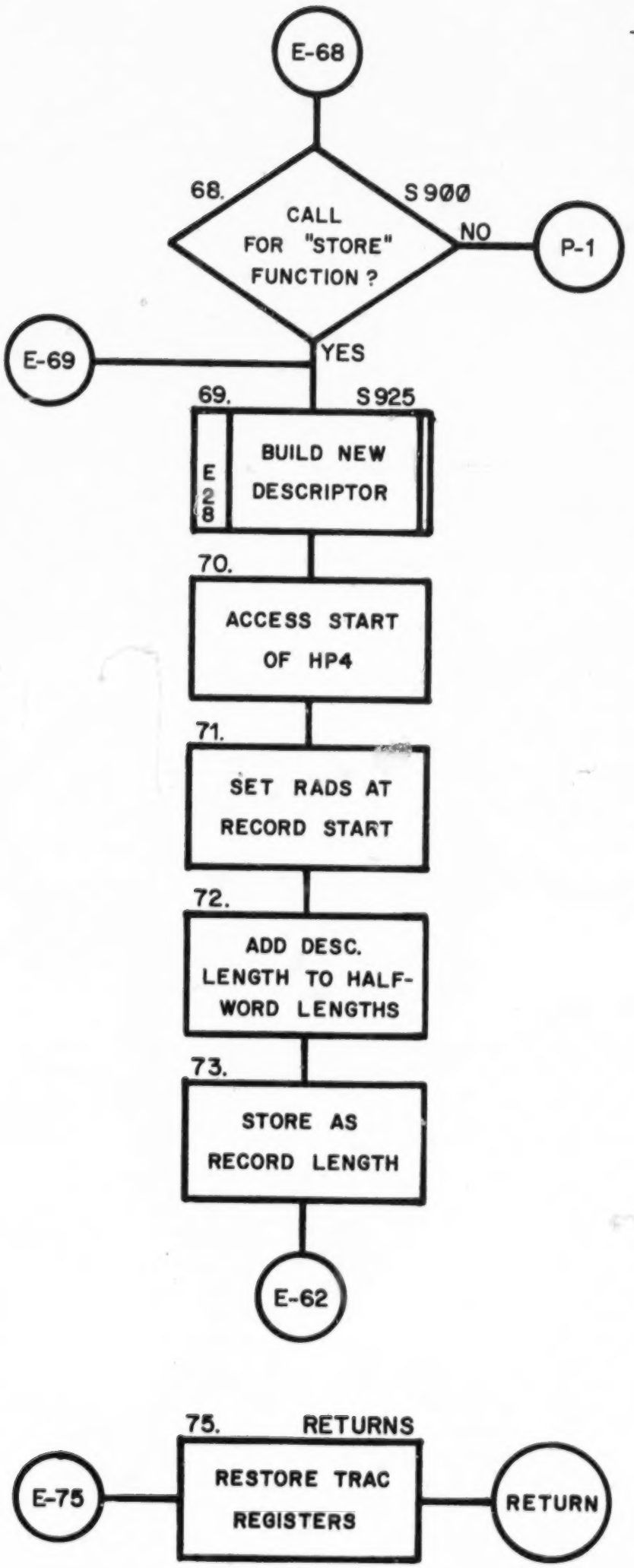

E-9
9.
WAS "INSERT" FUNCTION USED ?
YES
16.
S225
SET ERROR IN CALLER FPB: CALLING HASM
C-118
NO
10.
WAS "DELETE" FUNCTION USED ?
YES
E-33
NO
11.
0
1
POINTER TO VALUE STRING COUNT
12.
NEW STRING LONGER THAN OLD STRING?
NO
YES
13.
ONE BYTE LONGER ?
NO
E-22
YES
14.
IS OLD LENGTH ODD ?
YES
NO
E-22
18.
S250
NEW VALUE STRING COUNT TO DESCRIPTOR
19.
IS COUNT ZERO ?
YES
NO
20.
NEW VALUE STRING TO DESCRIPTOR
E-63

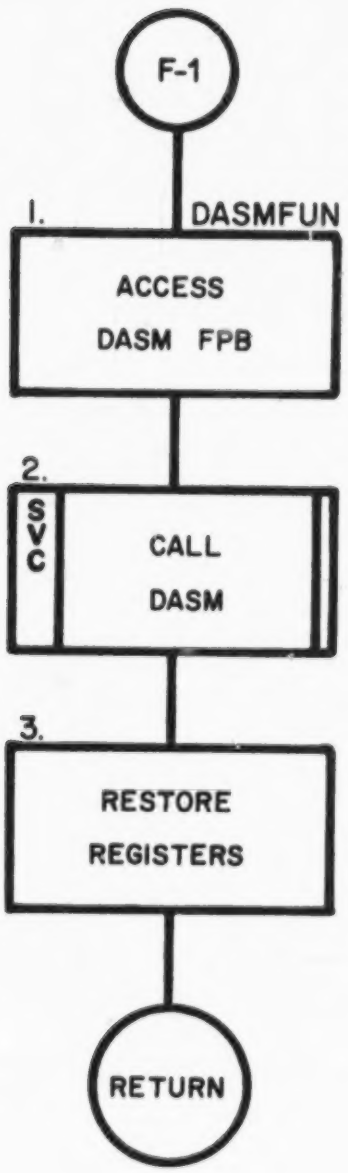

E-22
22.
S300
CAN NEW DESCRIPTOR FIT ?
NO
26.
S325
SET FPB ERROR: NO ROOM
C-118
YES
23.
SAME NAME ALREADY THERE?
YES
24.
NO
E 28
BUILD NEW DESCRIPTOR
E-54
28.
S350
MAKE FIRST HALF-WORD ZEROS
29.
STORE NAME COUNT AND NAME IN BUFFER
30
STORE VALUE STRING COUNT IN BUFFER
31.
IS DESCRIPTOR TOO LONG ?
YES
31A.
NO
MOVE TEXT OF VALUE STRING INTO BUFFER
RETURN

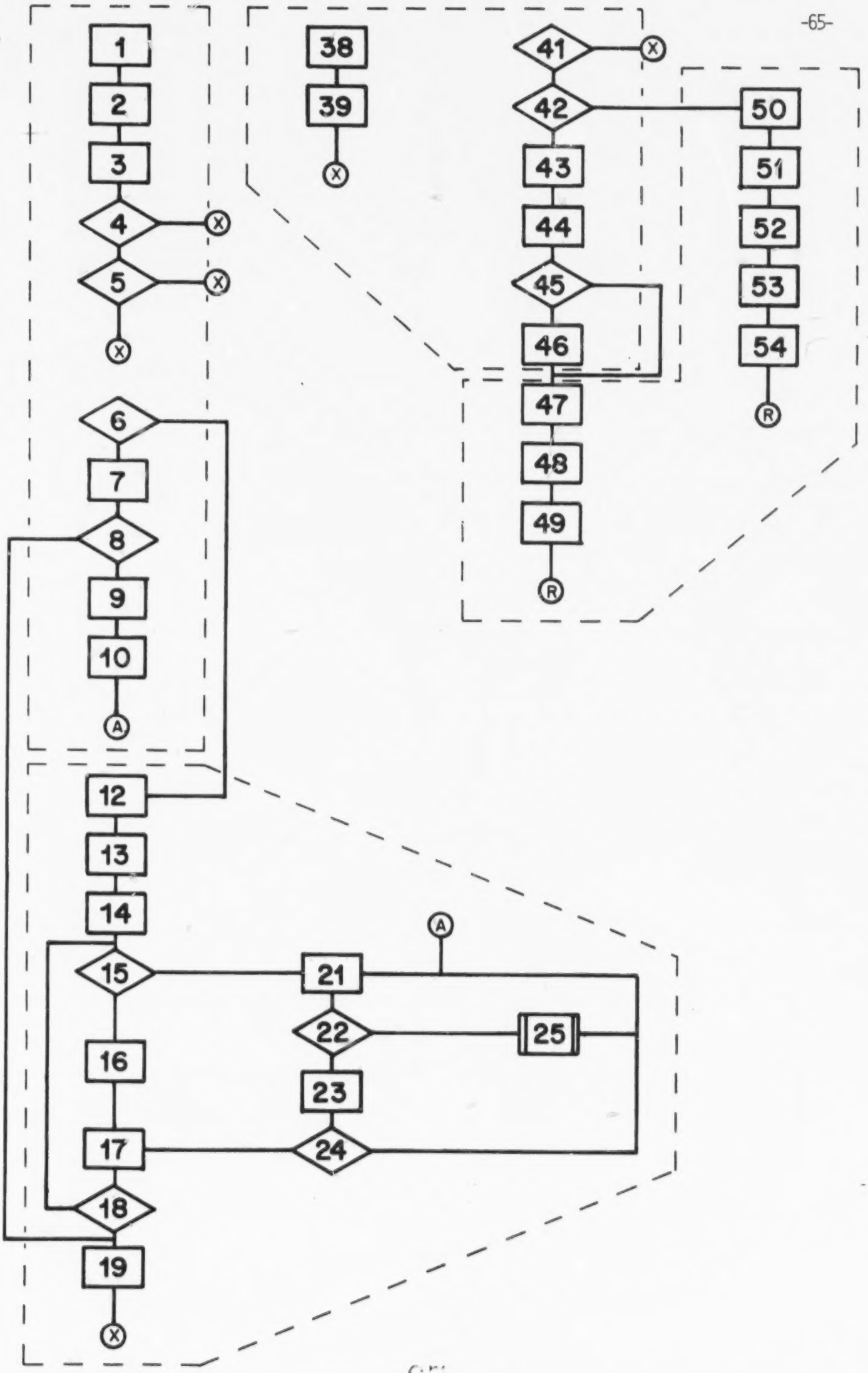

E-33
33.
EMPTY RECORD ?
S400
YES
E-52
NO
34.
ACCESS OLD DESCRIPTOR
35.
MOVE RAD OF OLD DESCRIPTOR
36.
OLD DESC. ORIGINALLY LAST DESC.?
NO
YES
37.
STORE ADDRESS OF PREVIOUS DESCRIPTOR
38.
CALL FOR "DELETE ONLY" FUNCTION ?
S500
YES
E-63
NO
39.
"DELETE WITH PRECEDING FETCH ?
YES
E-43
NO
40.
IS FIRST DESCRIPTOR RAD ZEROS ?
YES
E-69
NO
41.
E28
BUILD NEW DESCRIPTOR
E-54

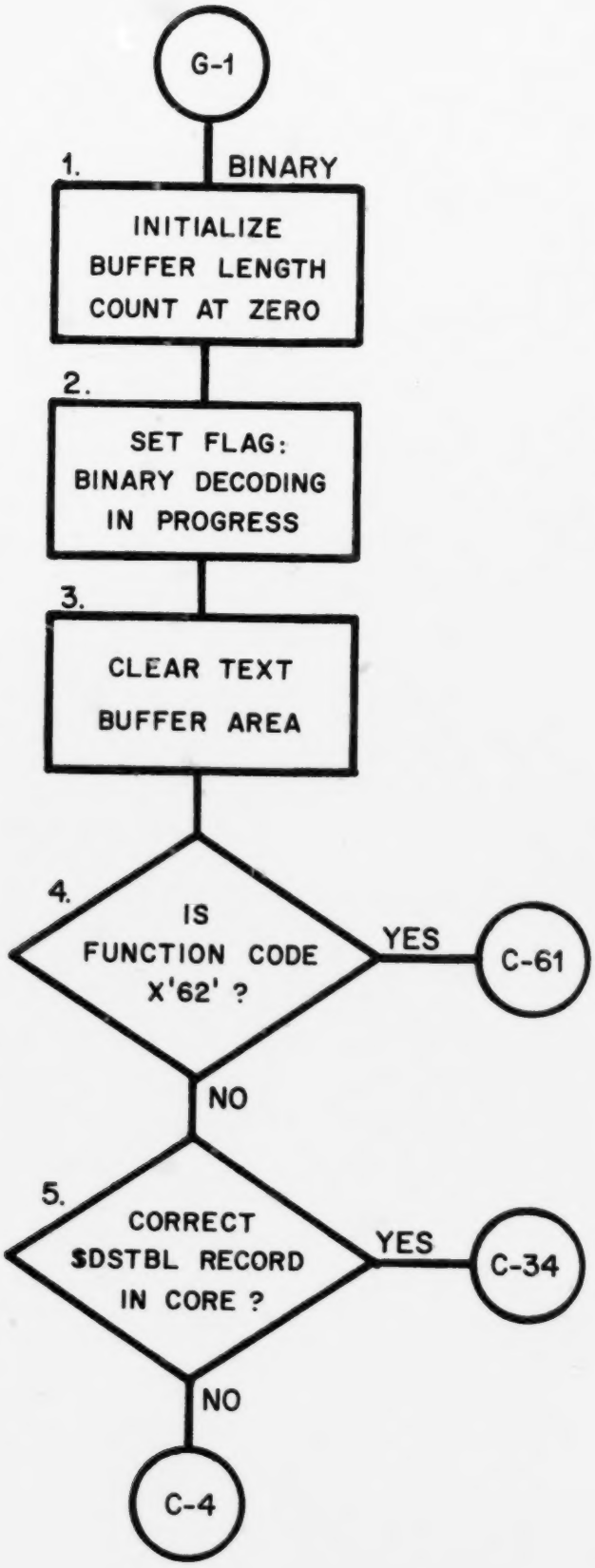

E-43
43.
DELFETED
E 2 8
BUILD NEW DESCRIPTOR
44.
MOVE VALUE STRING COUNT TO CALLER'S FPB
45.
MOVE VALUE STRING TEXT TO "DECODE" BUFFER
46.
STORE BUFFER ADDRESS IN CALLER'S FPB
E-63
E-48
48.
S700
CALL FOR "REPLACE DESC." FUNCTION ?
YES
NO
49.
ONE OF "DELETE" FUNCTIONS ?
YES
NO
50.
EMPTY OF VALID DESCRIPTORS ?
YES
E-69
NO
E-22
E-52
52.
S750
SET ERROR IN CALLER'S FPB
E-75

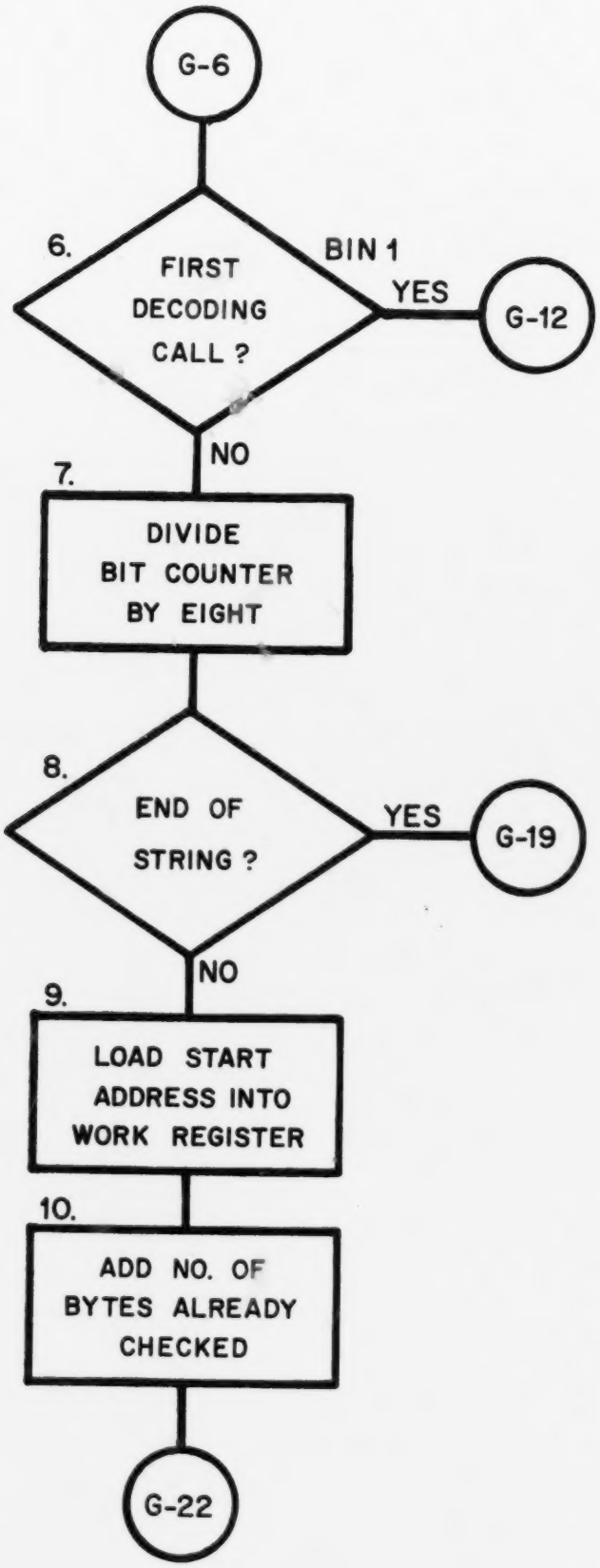

E-54
54.
S800
ACCESS RECORD START IN HP4
55.
ACCESS LAST RAD
56.
ADD RAD TO START ADDRESS
57.
UPDATE NEW RAD
58.
ALIGN RAD TO HALF-WORD BOUNDARY
59.
STORE RAD IN SECOND HALF-WORD
60.
ADJUST RECORD LENGTH
61.
NEW LENGTH TO DASM FPB
E-62
62.
S850
NEW DESCRIPTOR TO END OF RECORD
E-63
63.
S875
SET STORAGE FUNCTION REQUEST
64.
ZERO OUT ERROR INDICATOR
65.
SVC
CALL DASM
66.
"STORE" SUCCESSFUL ?
YES
E-75
NO
P-1

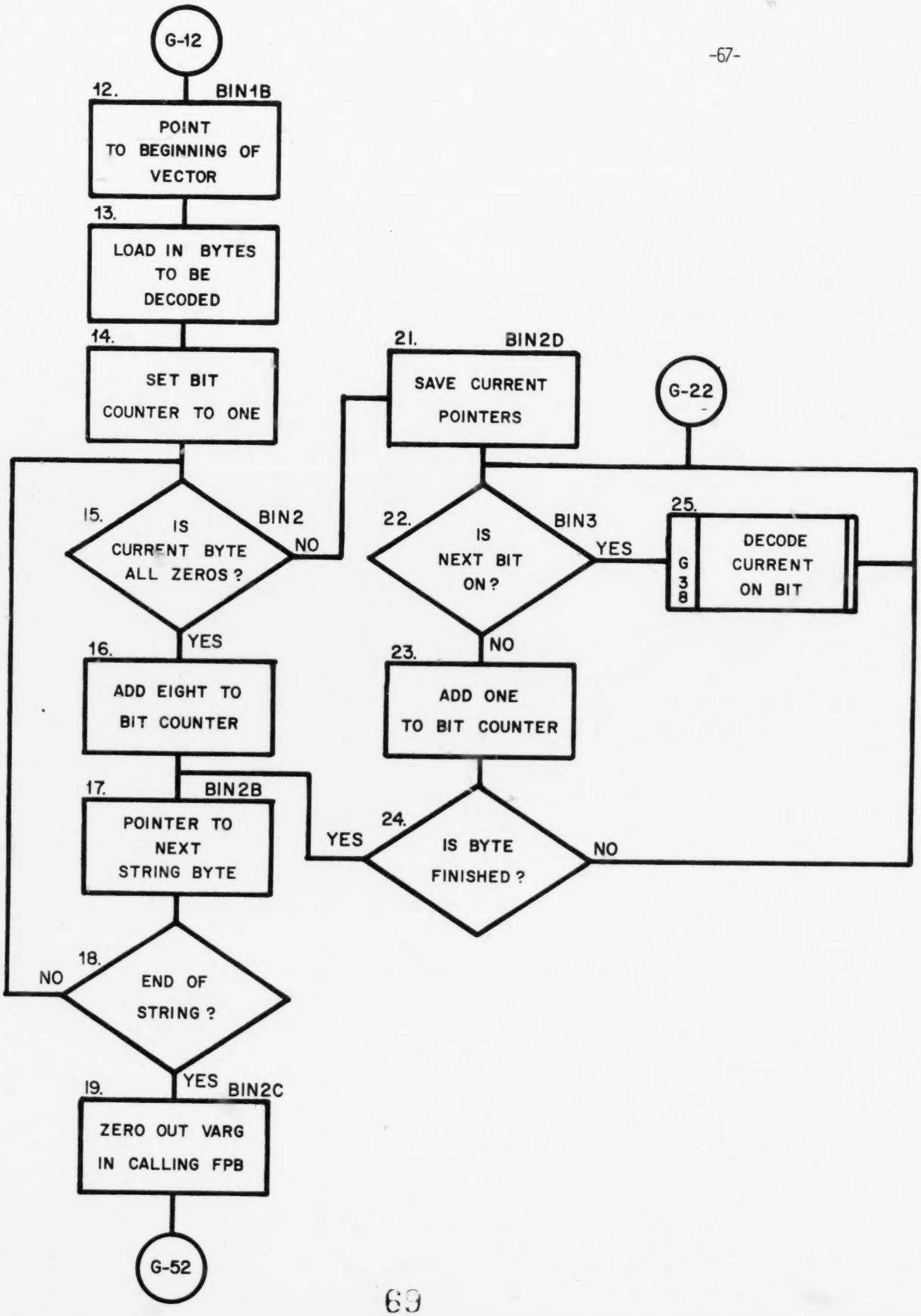

E-68
68.
CALL FOR "STORE" FUNCTION ?
S900
NO
P-1
YES
E-69
69.
S925
E28
BUILD NEW DESCRIPTOR
70.
ACCESS START OF HP4
71.
SET RADS AT RECORD START
72.
ADD DESC. LENGTH TO HALF-WORD LENGTHS
73.
STORE AS RECORD LENGTH
E-62
75.
RETURNS
E-75
RESTORE TRAC REGISTERS
RETURN

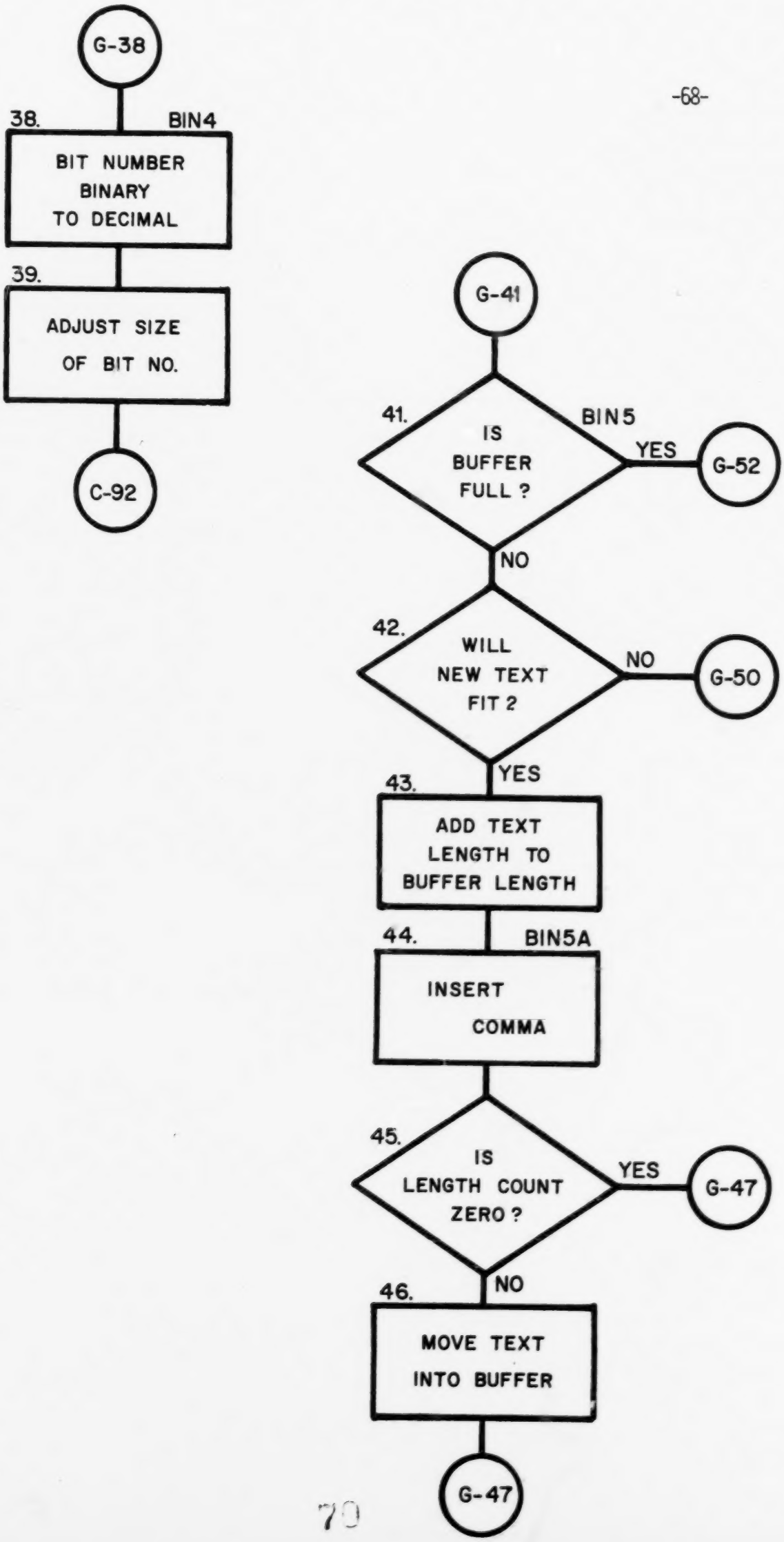

F-1
1.
DASMFUN
ACCESS
DASM FPB
2.
SVC
CALL
DASM
3.
RESTORE
REGISTERS
RETURN

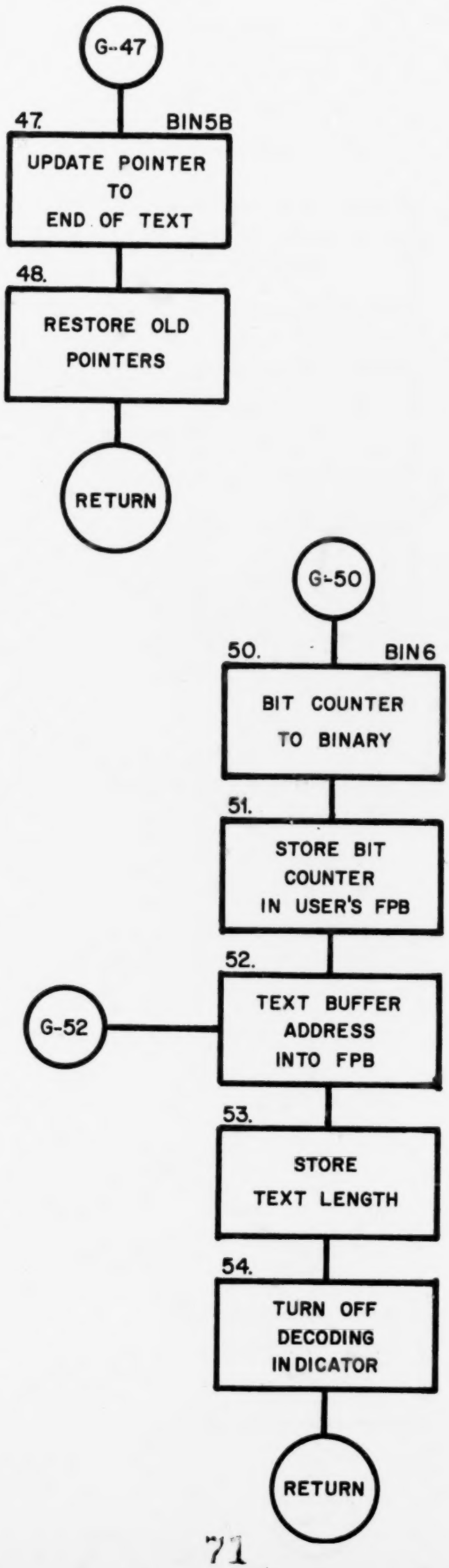

1
2
3
4
5
6
7
8
9
10
A
12
13
14
15
16
17
18
19
X
38
39
X
21
A
22
23
24
25
41
42
43
44
45
46
47
48
49
R
50
51
52
53
54
R

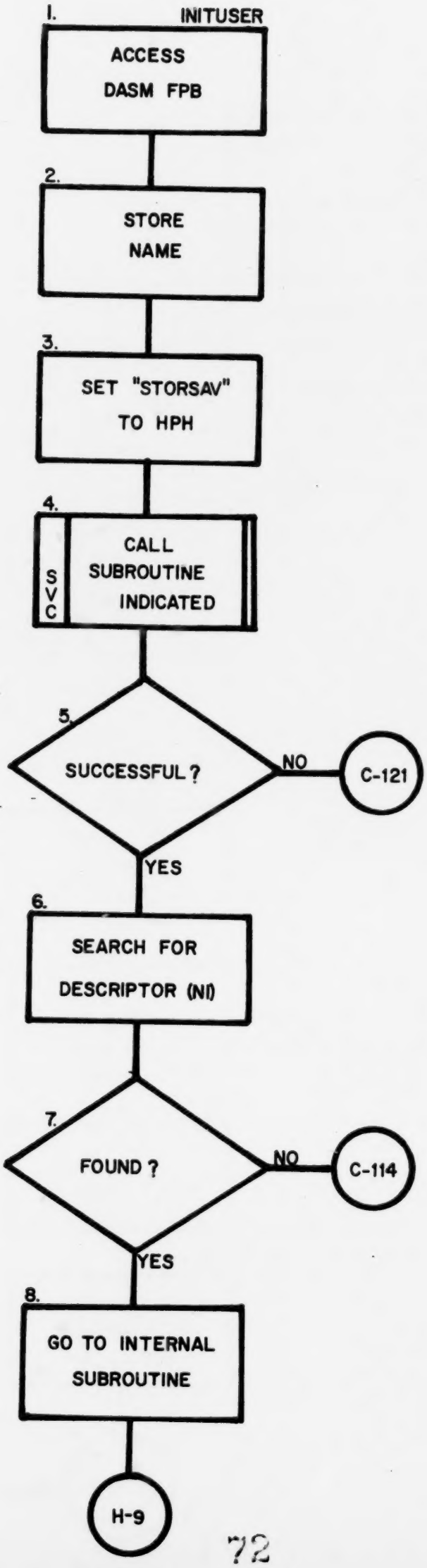

G-1
1.
BINARY
INITIALIZE BUFFER LENGTH COUNT AT ZERO
2.
SET FLAG: BINARY DECODING IN PROGRESS
3.
CLEAR TEXT BUFFER AREA
4.
IS FUNCTION CODE X'62' ?
YES
C-61
NO
5.
CORRECT $DSTBL RECORD IN CORE ?
YES
C-34
NO
C-4

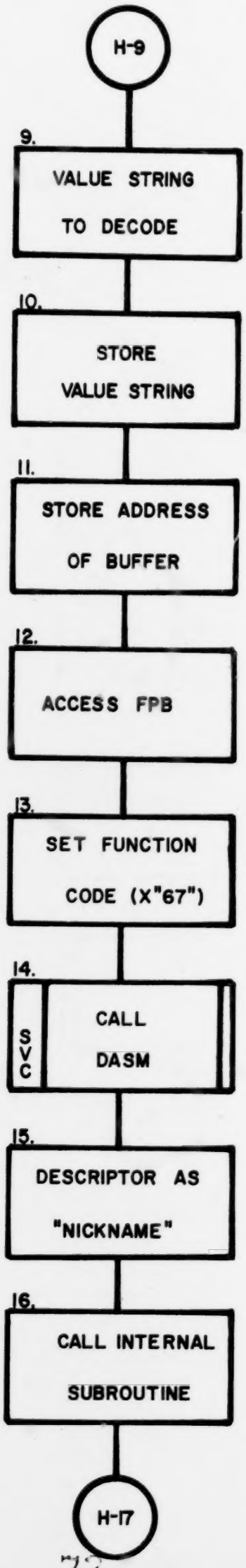

G-6
6.
FIRST DECODING CALL ?
BIN 1
YES
G-12
NO
7.
DIVIDE BIT COUNTER BY EIGHT
8.
END OF STRING ?
YES
G-19
NO
9.
LOAD START ADDRESS INTO WORK REGISTER
10.
ADD NO. OF BYTES ALREADY CHECKED
G-22

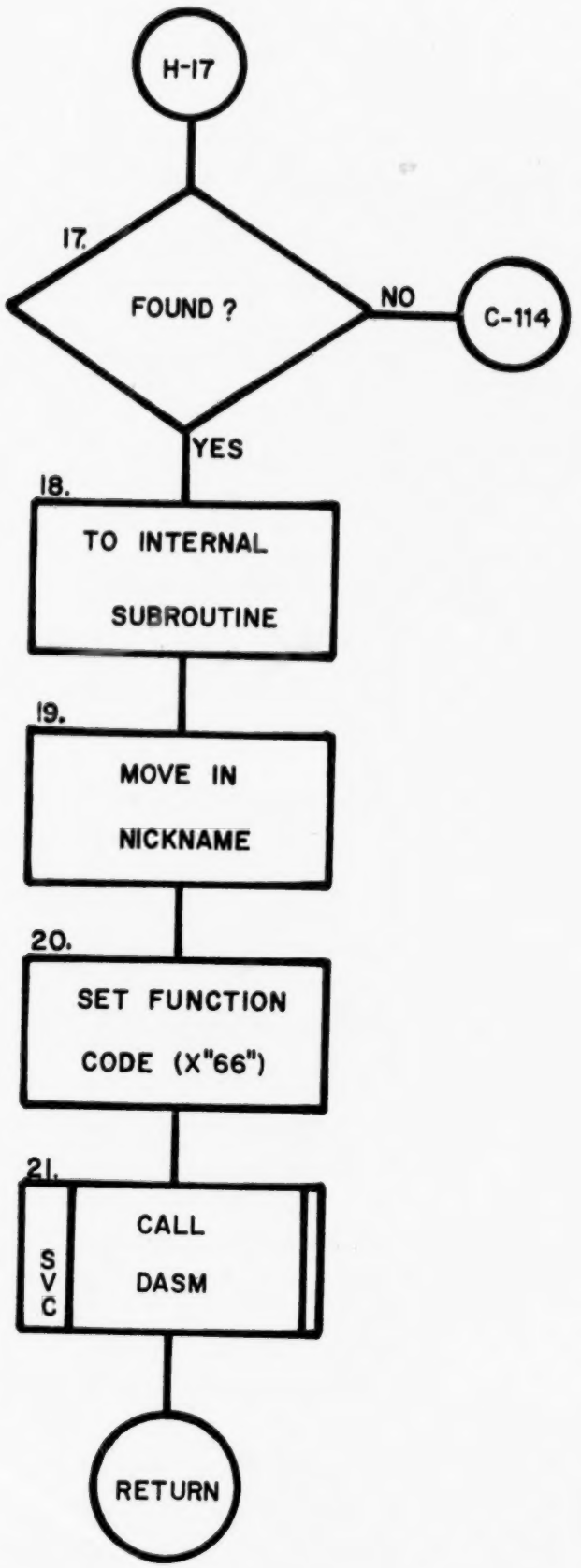

G-12
12.
BIN1B
POINT TO BEGINNING OF VECTOR
13.
LOAD IN BYTES TO BE DECODED
14.
SET BIT COUNTER TO ONE
15.
IS CURRENT BYTE ALL ZEROS ?
BIN2
NO
YES
16.
ADD EIGHT TO BIT COUNTER
17.
BIN2B
POINTER TO NEXT STRING BYTE
18.
NO
END OF STRING ?
YES
19.
BIN2C
ZERO OUT VARG IN CALLING FPB
G-52
21.
BIN2D
SAVE CURRENT POINTERS
G-22
22.
IS NEXT BIT ON ?
BIN3
YES
NO
23.
ADD ONE TO BIT COUNTER
24.
IS BYTE FINISHED ?
YES
NO
25.
G38
DECODE CURRENT ON BIT

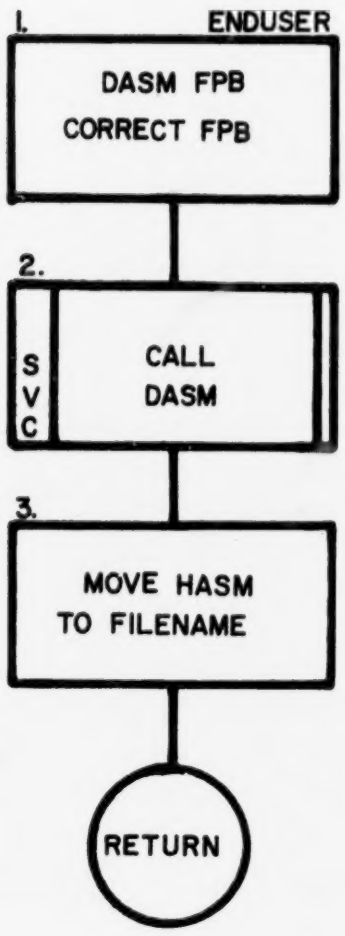

G-38
38.
BIN4
BIT NUMBER
BINARY
TO DECIMAL
39.
ADJUST SIZE
OF BIT NO.
C-92
G-41
41.
BIN5
IS
BUFFER
FULL ?
YES
G-52
NO
42.
WILL
NEW TEXT
FIT 2
NO
G-50
YES
43.
ADD TEXT
LENGTH TO
BUFFER LENGTH
44.
BIN5A
INSERT
COMMA
45.
IS
LENGTH COUNT
ZERO ?
YES
G-47
NO
46.
MOVE TEXT
INTO BUFFER
G-47

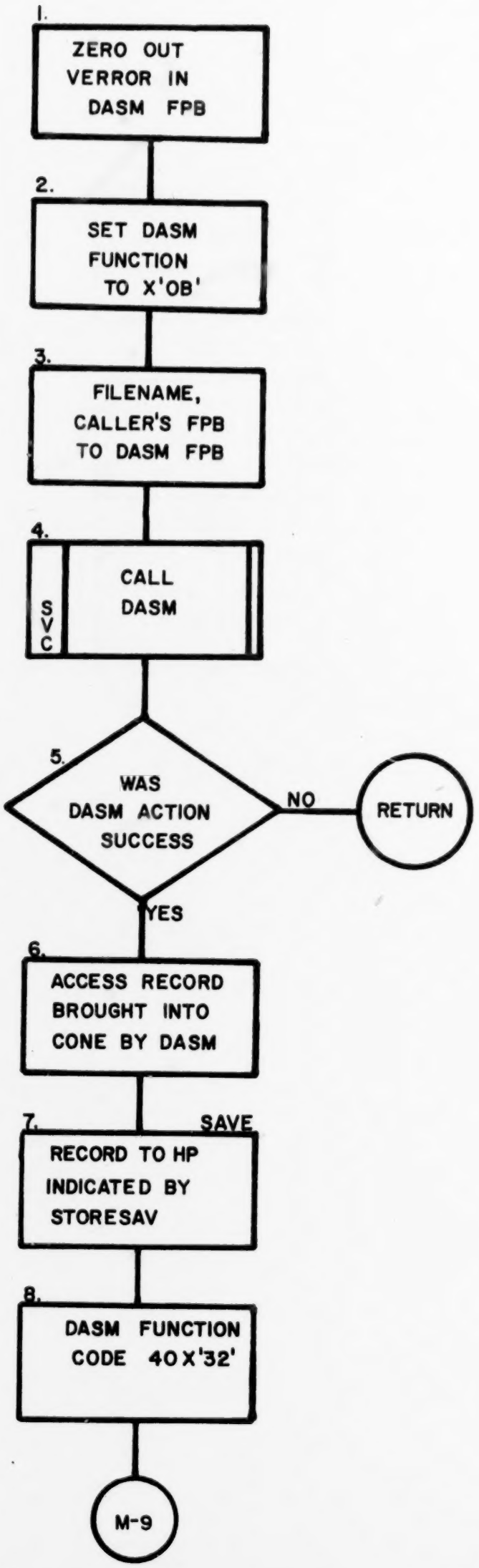

G-47
47.
BIN5B
UPDATE POINTER TO END OF TEXT
48.
RESTORE OLD POINTERS
RETURN
G-50
50.
BIN6
BIT COUNTER TO BINARY
51.
STORE BIT COUNTER IN USER'S FPB
52.
G-52
TEXT BUFFER ADDRESS INTO FPB
53.
STORE TEXT LENGTH
54.
TURN OFF DECODING INDICATOR
RETURN

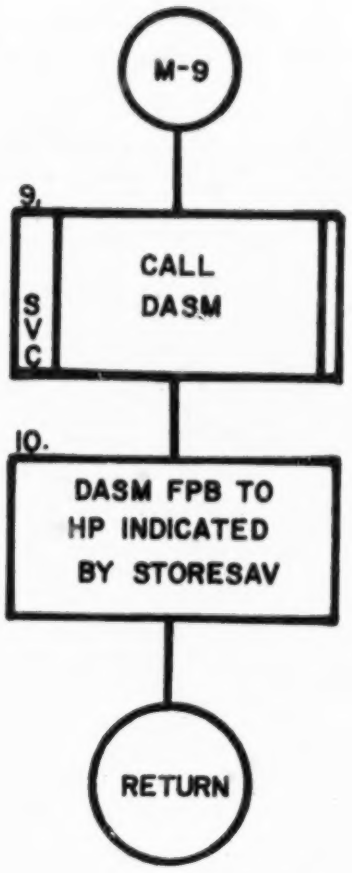

1.
INITUSER
ACCESS
DASM FPB
2.
STORE
NAME
3.
SET "STORSAV"
TO HPH
4.
SVC
CALL
SUBROUTINE
INDICATED
5.
SUCCESSFUL ?
NO
C-121
YES
6.
SEARCH FOR
DESCRIPTOR (NI)
7.
FOUND ?
NO
C-114
YES
8.
GO TO INTERNAL
SUBROUTINE
H-9

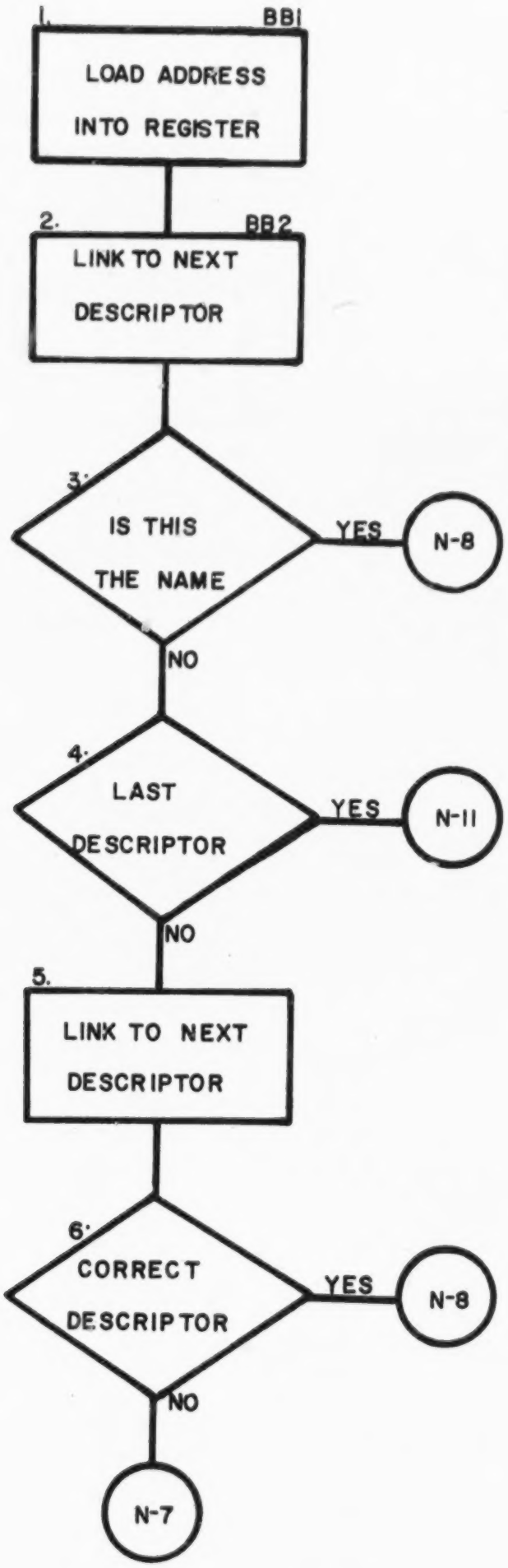

H-9
9.
VALUE STRING
TO DECODE
10.
STORE
VALUE STRING
11.
STORE ADDRESS
OF BUFFER
12.
ACCESS FPB
13.
SET FUNCTION
CODE (X"67")
14.
SVC
CALL
DASM
15.
DESCRIPTOR AS
"NICKNAME"
16.
CALL INTERNAL
SUBROUTINE
H-17

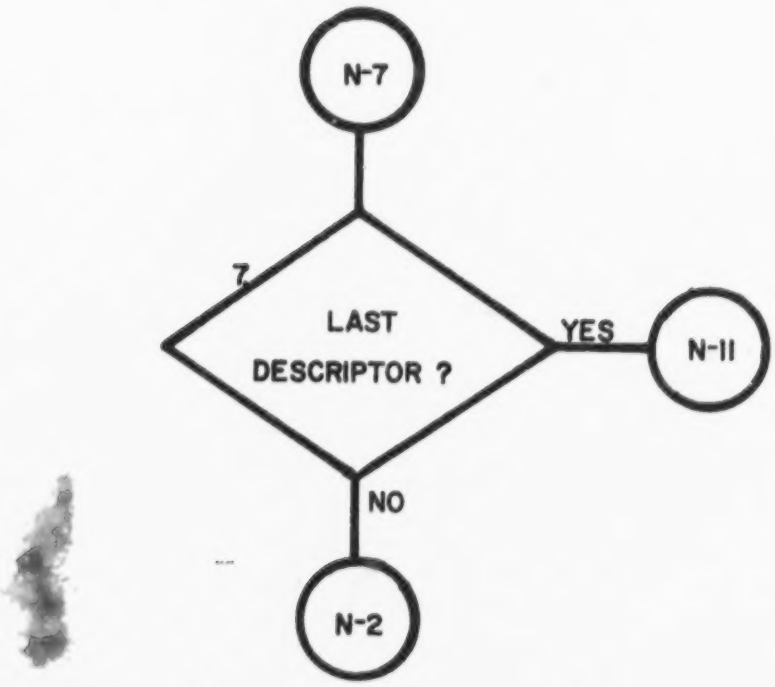

H-17
17.
FOUND ?
NO
C-114
YES
18.
TO INTERNAL
SUBROUTINE
19.
MOVE IN
NICKNAME
20.
SET FUNCTION
CODE (X"66")
21.
SVC
CALL
DASM
RETURN

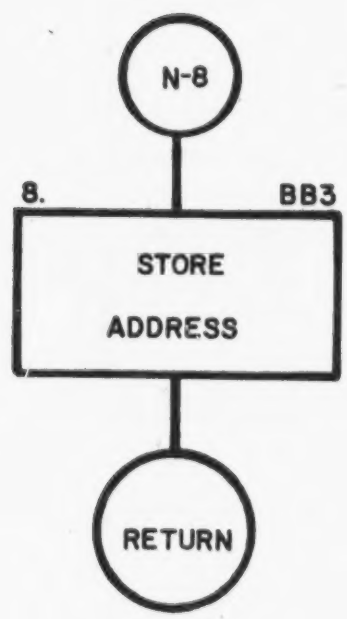

1.
ENDUSER
DASM FPB
CORRECT FPB
2.
SVC
CALL
DASM
3.
MOVE HASM
TO FILENAME
RETURN

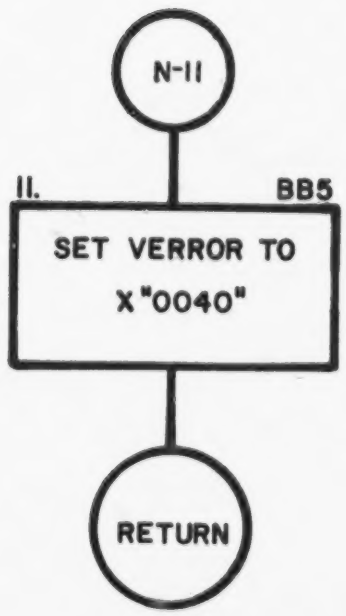

1.
ZERO OUT VERROR IN DASM FPB
2.
SET DASM FUNCTION TO X'0B'
3.
FILENAME, CALLER'S FPB TO DASM FPB
4.
SVC
CALL DASM
5.
WAS DASM ACTION SUCCESS
NO
RETURN
YES
6.
ACCESS RECORD BROUGHT INTO CONE BY DASM
7.
SAVE
RECORD TO HP INDICATED BY STORESAV
8.
DASM FUNCTION CODE 40X'32'
M-9

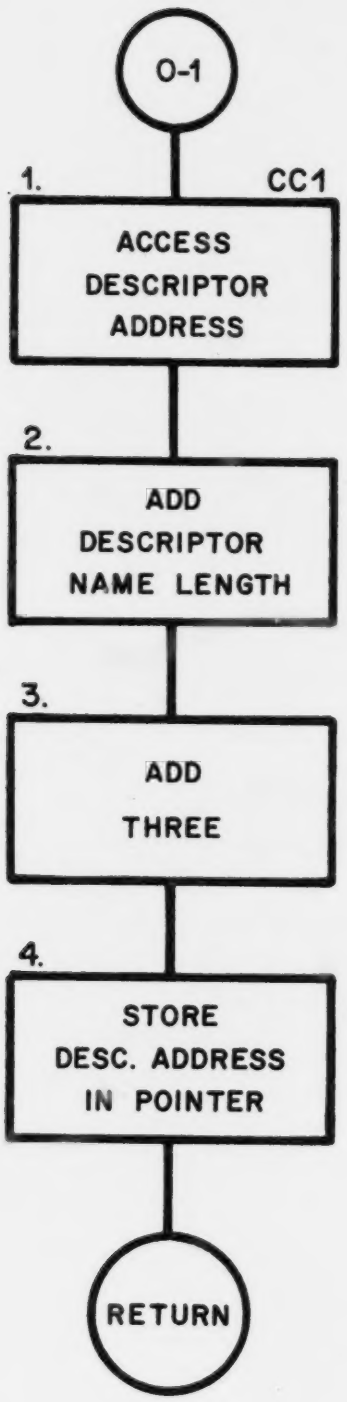

M-9
9.
SVC
CALL
DASM
10.
DASM FPB TO
HP INDICATED
BY STORESAV
RETURN


1.
BB1
LOAD ADDRESS
INTO REGISTER
2.
BB2
LINK TO NEXT
DESCRIPTOR
3.
IS THIS
THE NAME
YES
N-8
NO
4.
LAST
DESCRIPTOR
YES
N-11
NO
5.
LINK TO NEXT
DESCRIPTOR
6.
CORRECT
DESCRIPTOR
YES
N-8
NO
N-7


N-7
7.
LAST
DESCRIPTOR ?
YES
N-11
NO
N-2


N-8
8.
BB3
STORE
ADDRESS
RETURN


N-11
11.
BB5
SET VERROR TO
X"0040"
RETURN


0-1
1.
CC1
ACCESS
DESCRIPTOR
ADDRESS
2.
ADD
DESCRIPTOR
NAME LENGTH
3.
ADD
THREE
4.
STORE
DESC. ADDRESS
IN POINTER
RETURN